JFPDA 2008
Metz, France, 19-20 juin 2008

Supélec, Campus de Metz

Journées Francophones de Planification,
Décision et Apprentissage
pour la conduite de systèmes

Supélec

Introduction

Les méthodes d'optimisation de décisions séquentielles sont nombreuses et font l'objet de recherches dans différents domaines, notamment en intelligence artificielle. Dans ce cadre, l'apprentissage par renforcement, les processus décisionnels de Markov, l'ordonnancement, ou la planification offrent des réponses particulières au problème général de la recherche de politiques ou stratégies de commande optimale. Les Journées Francophones de Planification, Décision et Apprentissage pour la conduite des systèmes (JFPDA) ont pour but de rassembler la communauté de chercheurs francophones en intelligence artificielle travaillant sur ces problèmes. En particulier, elles offriront un espace de rencontre entre des chercheurs confirmés et les doctorants et post-doctorants présentant éventuellement des travaux en cours.

Liste non exhaustive des thématiques abordées :

- apprentissage par renforcement ;
- processus décisionnels de Markov, totalement ou partiellement observables, factorisés ou hiérarchiques ;
- planification ;
- ordonnancement ;
- programmation dynamique ;
- généralisation ;
- approximation de fonction de valeur ;
- recherche directe de politique ;
- contrôle de systèmes continus ou hybrides ;
- systèmes mono- ou multi-agents ;
- gestion de l'incertitude ;
- approches d'inspiration biologique.

Le comité d'organisation :

Olivier PIETQUIN
Hervé FREZZA-BUET

Comité D'organisation

Olivier PIETQUIN (Olivier.Pietquin@supelec.fr)
Organisation Scientifique

Équipe Information, Multimodalité & Signal
Supélec, Campus de Metz
2 rue Edouard Belin, 57070 Metz (France)

Hervé FREZZA-BUET (Hervé.Frezza-Buet@supelec.fr)
Organisation Scientifique

Équipe Information, Multimodalité & Signal
Supélec, Campus de Metz
2 rue Edouard Belin, 57070 Metz (France)

Claudine MERCIER (Claudine.Mercier@supelec.fr)
Site Internet

Supélec, Campus de Metz
2 rue Edouard Belin, 57070 Metz (France)

Patrick MERCIER (Patrick.Mercier@supelec.fr)
Moyens Techniques

Équipe Information, Multimodalité & Signal
Supélec, Campus de Metz
2 rue Edouard Belin, 57070 Metz (France)

Thérèse FRESSENGEAS (Therese.Fressengeas@supelec.fr)
Administration

Supélec, Campus de Metz
2 rue Edouard Belin, 57070 Metz (France)

Comité de Programme

Alain Dutech (Loria, Nancy)
Bruno Scherrer (Loria, Nancy)
Olivier Sigaud (AnimatLab, Paris)
Olivier Buffet (Loria, Nancy)
François Charpillet (Loria, Nancy)
Gérard Verfaillie (ONERA, Toulouse)
Thierry Vidal (IRISA, Rennes)
Humbert Fiorino (IMAG, Grenoble)
Damien Pellier (Paris Descartes, Paris)
Philippe Mathieu (LIFL, Université de Lille)
Olivier Pietquin (Supélec, campus de Metz)
Hervé Frezza-Buet (Supélec, campus de Metz)

Liste des Participants

AKPLOGAN	Mahuna	INRIA / LORIA
ARAS	Raghav	INRIA / LORIA
BELAIDI	Ilias	Université Paris V
BIDOT	Julien	Ulm University
BOUGHZANE	Jalel	
BOUSSARD	Matthieu	Université de Caen
BUFFET	Olivier	INRIA / LORIA
CHANONI	Emilie	Université de Rouen
CHARPILLET	François	INRIA / LORIA
CORONAA	Gabriel	INRIA / LORIA
DIBANGOYE	Gilles	Université de Caen
DUTECH	Alain	INRIA / LORIA
EL FALOU	Mohamad	Université de Caen
FIORINO	Humbert	LIG GRENOBLE
FREZZA-BUET	Hervé	SUPELEC
FRICOUT	Gabriel	ARCELOR MITTAL
GABILLON	Yoann	Laboratoire Informatique de Grenoble
GEIST	Matthieu	ARCELOR MITTAL
GUITTON	Julien	ONERA
HOOK	Jean-Baptiste	INRIA
HREN	Jean-François	INRIA
KLEIN	François	INRIA / LORIA
KOZLOVA	Olga	THALES
LENGAGNE	Sébastien	INRIA
PAUCHET	Alexandre	LITIS - INSA de Rouen
PELLIER	Damien	Université Paris V
PIETQUIN	Olivier	SUPELEC
RACHELSON	Emmanuel	ONERA
RIMMEL	Arpad	INRIA
ROSE	Cédric	INRIA / LORIA
SCHERRER	Bruno	INRIA / LORIA
SIGAUD	Olivier	ISIR - UPMC
STEFANOVITCH	Nicolas	
TEYTAUD	Olivier	INRIA
THIERY	Christophe	INRIA / LORIA
VERFAILLE	Gérard	ONERA

Programme

Jeudi 19 juin 2008

9h - 9h45	Accueil
9h45 - 10h	Ouverture

Session 1 : Planification et Ordonnancement I

10h - 10h30	*Utiliser des chronogrammes pour modéliser des problèmes de planification et d'ordonnancement* Gérard VERFAILLIE et Cédric PRALET
10h30 - 11h	*Vers un modèle de planification hybride pour la planification d'action et de déplacement* Julien GUITTON et Jean-Loup FARGES
11h - 11h30	Pause café
11h30 - 12h	*Including expert knowledge in Bandit-Based Monte-Carlo planning, with application to Computer-Go* Olivier TEYTAUD, Sylvain GELLY, Jean Baptiste HOOCK, Julien PEREZ et Arpad RIMMEL
12h00 - 12h30	*Méthode pour la Planification de Trajectoires Garanties* Sébastien LENGAGNE, Nacim RAMDANI et Philippe FRAISSE

Repas

Session 2 : MDP et Programmation Dynamique I

14h - 14h30	*Vector Valued Markov Decision Process for robot platooning* Matthieu BOUSSARD, Maroua BOUZID and Abdel-Illah MOUADDIB
14h30-15h	*Une solution pour le probleme de Contrôle de Troupeau de Moutons avec un modèle DEC-MDP* Jalel BOUGHIZANE, Narjes DOGGAZ and Mourad SAADANA
15h - 15h30	Pause café
15h30 - 16h	*Topological Order Based Planning for Solving POMDPs* Jilles Steeve DIBANGOYE, Brahim CHAIB-DRAA et Abdel-Illah MOUADDIB
16h-16h30	*Une nouvelle heuristique pour l'alignement de motifs 2D par programmation dynamique* Emilie CHANONI, Thierry LECROQ et Alexandre PAUCHET

Discussion et organisation des prochaines journées

Vendredi 20 juin 2008

Session 3 : MDP et Programmation Dynamique II

9h - 9h30	*Apprentissage par renforcement hiérarchique dans les MDP factorisés* Olga KOZLOVA, Olivier SIGAUD et Christophe MEYER
9h30 - 10h	*Un Algorithme Amélioré d'Itération de la Politique Approchée pour les Processus Décisionnels Semi-Markoviens Généralisés* Emmanuel RACHELSON, Patrick FABIANI et Frédérick GARCIA
10h - 10h30	*Filtrage bayésien de la récompense* Matthieu GEIST, Olivier PIETQUIN et Gabriel FRICOURT
10h30 - 11h	*Optimistic planning of deterministic system* Jean Francois HREN et Remi MUNOS
11h - 11h30	Pause café

Session 4 : Planification et Ordonnancement II

11h30 - 12h	*Planification distribuée par fusions incrémentales de graphes* Damien PELLIER et Ilias BELAIDI
12h - 12h30	*Composition automatique et dynamique de Services Web utilisant des techniques de planification* Mohamad EL FALOU, Maroua BOUZID, Thierry VIDAL et Abdel-Illah MOUADDIB
12h30 - 13h	*Composition dynamique de systèmes interactifs : l'alliance de l'IHM et de la Planification* Yoann GABILLON, Gaëlle CALVARY et Humbert FIORINO

Repas

Utiliser des chronogrammes pour modéliser des problèmes de planification et d'ordonnancement

Gérard Verfaillie and Cédric Pralet

ONERA/DCSD, 2 av. Édouard Belin, BP 74025, 31055, Toulouse Cédex 4, France
Gerard.Verfaillie@onera.fr, Cedric.Pralet@onera.fr

Résumé :
Le cadre CNT (*Constraint Network on Timelines* (Verfaillie *et al.*, 2008)) a été conçu pour modéliser des systèmes dynamiques à événements discrets et les propriétés que l'on connaît, que l'on veut vérifier ou que l'on veut imposer sur de tels systèmes. Dans cet article, après un rappel sur le cadre CNT, nous montrons deux exemples concrets de modélisation de problèmes de planification et d'ordonnancement issus du domaine aérospatial : les problèmes de gestion de la mission pour une patrouille de drones, d'une part, et pour un satellite de surveillance et d'observation de la Terre, d'autre part.
Mots-clés : Planification, Ordonnancement, Chronogrammes, Contraintes.

1 Le cadre CNT

Le cadre CNT (*Constraint Network on Timelines* (Verfaillie *et al.*, 2008), *Réseau de Contraintes sur des Chronogrammes* en français) est un cadre générique de modélisation de systèmes dynamiques à événements discrets à base de *variables* et de *contraintes*, telles qu'elles sont définies dans le cadre CSP (*Constraint Satisfaction Problem* (Rossi *et al.*, 2006)). Plus précisément, le cadre CNT permet de modéliser tout système dynamique à événements discrets (incluant sa dynamique et les propriétés que l'on désire imposer ou vérifier sur elle) comme un *CSP dynamique* (Mittal & Falkenhainer, 1990), aussi dénommé *CSP conditionnel*, c'est-à-dire un CSP dont l'ensemble des variables et des contraintes n'est pas fixé a priori, mais dépend de l'affectation de variables particulières, appelées variables de *dimension*.

Le cadre CNT est bâti sur l'hypothèse classique d'*étapes* discrètes auxquelles des *événements* et des *changements d'état* instantanés se produisent. Ses composants de base sont des *attributs* qui peuvent être de trois types : *temps*, *état* ou *événement*. À chaque attribut, est associé, en plus de son type, un *domaine* de *valeurs* possibles qui peut être discret ou continu, fini ou infini. Les attributs de type *temps* permettent de représenter les positions temporelles des étapes successives. Les attributs de type *état* permettent de représenter les valeurs que prennent les composantes permanentes du système lors des étapes successives (composantes permanentes dans le sens où elles sont tout le temps présentes, même si leur valeur change étape après étape, comme par exemple un niveau de ressource). Les attributs de type *événement* permettent enfin de représenter les valeurs des phénomènes instantanés du système lors des mêmes étapes successives. La seule différence entre attributs de type *état* et attributs de type *événement* est l'hypothèse qu'entre deux étapes successives un attribut de type *état* garde la valeur qu'il a prise lors de la première des deux étapes (hypothèse d'évolution constante par morceaux) tandis qu'un attribut de type *événement* ne prend aucune valeur (ou une valeur par défaut représentant l'absence d'événement). Voir les figures 1 et 2.

Un *chronogramme* permet de réunir des attributs complètement synchronisés, qui partagent la même séquence d'étapes, avec les mêmes positions temporelles. À chaque chronogramme, sont associés (i) une *variable*, dite de *dimension*, qui représente le nombre d'étapes du chronogramme, (ii) un *ensemble* fini d'*attributs* de type *état* ou *événement* et (iii) au plus un *attribut* de type *temps* (aucun si les positions temporelles des étapes successives ne sont pas pertinentes). Le domaine de la variable de dimension peut être fini ou infini. À chaque chronogramme, à chaque étape et à chaque attribut du chronogramme, est associée une *variable*, dite d'*attribut*, qui représente la valeur de cet attribut à cette étape. Différents chronogrammes permettent de représenter des évolutions concurrentes, éventuellement partiellement synchronisées.

Un *réseau de contraintes sur des chronogrammes* (CNT) est défini par (i) un *ensemble* de *chronogrammes* qui définit lui-même un *ensemble* de *variables* (une variable de dimension par chronogramme et une variable d'attribut par attribut et par étape dans le chronogramme associé) et (ii) par un *ensemble* de *contraintes*

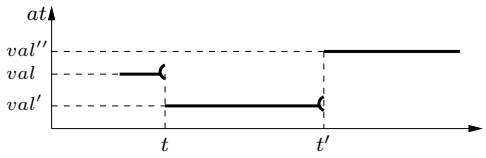


FIGURE 1 – Évolution d'un attribut d'état.

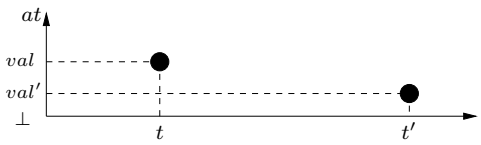


FIGURE 2 – Évolution d'un attribut d'événement.

sur ces variables (sans aucune limitation sur les variables impliquées, ni sur le type de contrainte liant ces variables). Cependant, à la différence des CSP classiques, l'ensemble de variables et de contraintes associés à un CNT n'est pas fixé a priori. Il dépend de la valeur prise par les variables de dimension. Le résultat est une forme de *CSP dynamique* (Mittal & Falkenhainer, 1990), aussi dénommé *CSP conditionnel*.

Plus formellement, on adopte les définitions suivantes.

Définition 1
Un attribut at est une paire $\langle ty(at), do(at)\rangle$, où $ty(at)$ est le type de at avec trois valeurs possibles ($state$, $event$ ou $time$) et $do(at)$ est le domaine de valeurs de at.

Si $ty(at) = state$, il n'y a aucune hypothèse sur la nature de $do(at)$: $do(at)$ peut être discret ou continu, fini ou infini. Si $ty(at) = event$, il n'y a aucune hypothèse non plus, exceptée la présence d'une valeur spéciale $\perp$ représentant l'absence d'événement. Finalement, si $ty(at) = time$, $do(at) \subseteq \mathbb{R}$, mais $do(at)$ peut être aussi discret ou continu, fini ou infini.

Définition 2
Un chronogramme tl est une paire $\langle ats(tl), ns(tl)\rangle$, où $ats(tl)$ est l'ensemble fini des attributs associés à tl et $ns(tl)$ est une variable représentant le nombre d'étapes de tl

On fait l'hypothèse qu'un chronogramme tl contient au plus un attribut de type $time$, noté $ti(tl)$ s'il existe. On fait aussi l'hypothèse qu'un attribut at appartient à un chronogramme et un seul, noté $tl(at)$. Pour un attribut at de type $state$ ou $event$, on note $ti(at) = ti(tl(at))$ l'attribut de type $time$ du chronogramme auquel il appartient, si un tel attribut existe. Finalement, on fait l'hypothèse que le domaine de valeurs $do(ns(tl))$ de la variable de dimension d'un chronogramme tl est un sous ensemble de $\mathbb{N}$, fini ou infini. Un domaine infini permet de représenter des évolutions non bornées. Les différentes étapes d'un chronogramme tl sont numérotées de 1 à $ns(tl)$. Pour un attribut at, on note $ns(at) = ns(tl(at))$ la variable de dimension du chronogramme auquel il appartient. À chaque chronogramme tl, chaque attribut at de tl et chaque étape $i \,|\, 1 \leq i \leq ns(tl)$, on associe une variable d'attribut at_i de domaine $do(at)$. Pour un attribut ti de type $time$, on fait de plus l'hypothèse que $\forall i \,|\, 2 \leq i \leq ns(tl)\,,\ ti_{i-1} \leq ti_i$ (les étapes successives sont temporellement totalement ordonnées) et, pour un attribut at de type $state$ ou $event$, on fait l'hypothèse que $\forall i \,|\, 2 \leq i \leq ns(tl)\,,\ (ti(at)_{i-1} = ti(at)_i) \rightarrow (at_{i-1} = at_i)$ (si deux étapes sucessives sont temporellement identiques, elles sont identiques sur l'ensemble des attributs).

Si l'on considère maintenant un ensemble fini tls de chronogrammes, on peut classer les variables associées à tls en variables de *dimension* (une par chronogramme) et variables d'*attribut* (pour chaque chronogramme, une par attribut et par étape du chronogramme). On note $vars(tls) =$ *d-vars*$(tls) \cup$ *a-vars*(tls). On peut aussi les classer différemment en variables *obligatoires* (ou *statiques*) et variables *optionnelles* (ou *dynamiques*). Les variables obligatoires sont les variables de dimension et les variables d'attribut associées aux étapes obligatoires dans chaque chronogramme, obligatoires du fait du minimum du domaine des variables de dimension (si, par exemple $do(ns(tl)) = [3..+\infty]$, le chronogramme tl a au moins trois étapes et, pour chaque attribut de tl, les variables d'attribut associées à ces trois étapes sont obligatoires). Les variables optionnelles sont les variables d'attribut non obligatoires. On note $vars(tls) =$ *m-vars*$(tls) \cup$ *o-vars*(tls) (m

pour *mandatory* et o pour *optional*). À noter que l'ensemble des variables optionnelles peut être infini, si le domaine de certaines variables de dimension n'est pas borné. Chaque affectation des variables de dimension de tls définit cependant un ensemble fini de variables optionnelles à considérer. Pour chaque affectation A des variables de dimension de tls, on note *o-vars*(tls, A) l'ensemble fini des variables optionnelles de tls résultant de l'affectation A.

Définition 3
Une affectation d'un ensemble fini tls de chronogrammes est une affectation de toutes les variables obligatoires de tls (m-vars(tls)) et de toutes les variables optionnelles résultant de l'affectation des variables de dimension de tls (o-vars(tls, A), si A est l'affectation des variables de dimension, qui constituent un sous-ensemble des variables obligatoires).

Définition 4
Une contrainte obligatoire (ou statique) c sur un ensemble fini tls de chronogrammes est une contrainte CSP classique, c'est-à-dire une paire $\langle sco(c), rel(c)\rangle$, où $sco(c)$ (la portée de c) est un sous-ensemble (fini) des variables obligatoires de tls ($sco(c) \subseteq$ m-vars(tls)) et $rel(c)$ (la relation associée à c) est n'importe quelle représentation explicite ou implicite des combinaisons de valeur autorisées pour les variables de $sco(c)$.

Définition 5
Une contrainte optionnelle (ou dynamique) c sur un ensemble fini tls de chronogrammes est une paire $\langle tls(c), fct(c)\rangle$, où $tls(c)$ est sous-ensemble (fini) des chronogrammes ($tls(c) \subseteq tls$) et $fct(c)$ est une fonction qui associe à chaque affectation A des variables de dimension de $tls(c)$ un ensemble fini de contraintes CSP classiques dont la portée est, pour chacune, un sous-ensemble (fini) de m-vars$(tls) \cup$ o-vars$(tls(c), A)$, l'ensemble des variables obligatoires de tls et des variables optionnelles de $tls(c)$ résultant de l'affectation A des variables de dimension de $tls(c)$.

Définition 6
Un réseau de contraintes cnt sur des chronogrammes est une paire $\langle tls(cnt), cs(cnt)\rangle$, où $tls(cnt)$ est un ensemble fini de chronogrammes et $cs(cnt)$ est un ensemble fini de contraintes obligatoires ou optionnelles sur $tls(cnt)$.

Définition 7
Une affectation d'un réseau de contraintes cnt sur des chronogrammes est une affectation de $tls(cnt)$. Elle est cohérente si et seulement si toutes les contraintes de $cs(cnt)$ sont satisfaites.

Définition 8
Un réseau de contraintes cnt sur des chronogrammes est cohérent si et seulement si il en existe une affectation cohérente.

La généricité du cadre CNT lui permet d'englober des cadres de modélisation classiques de systèmes dynamiques à événements discrets tels que les automates, les produits synchronisés d'automates, les automates temporisés et les réseaux de Petri, ainsi que les requêtes classiques sur ces cadres, telles que par exemple l'atteignabilité d'un état. Voir (Verfaillie *et al.*, 2008) pour les preuves associées.

Dans cet article, nous nous intéressons essentiellement à des problèmes de planification et d'ordonnancement. Avant de montrer comment des problèmes de gestion mission pour une patrouille de drones ou un satellite de surveillance et d'observation peuvent être modélisés dans le cadre CNT, nous commençons par des problèmes génériques largement connus qui illustrent les capacités de modélisation du cadre CNT : le problème de planification dans le cadre STRIPS et le problème de gestion de projets avec contraintes de ressources (RCPSP).

2 Modélisation du problème de planification dans le cadre STRIPS

Un problème de planification dans le cadre STRIPS (Fikes & Nilsson, 1971; Ghallab *et al.*, 2004) est défini par un quadruplet $\langle F, A, I, G\rangle$, où :

1. F est un ensemble fini de *fluents* à valeurs booléennes ;
2. A est un ensemble fini d'*actions*, avec chaque action $a \in A$ définie par un triplet $\langle p_a, e_a^+, e_a^-\rangle \mid p_a, e_a^+, e_a^- \subseteq F$, où p_a, e_a^- et e_a^+ définissent respectivement les *pré-conditions*, les *effets positifs* et les *effets négatifs* de a ;

3. $I \subseteq F$ définit l'*état initial* ;
4. G est un ensemble de conditions logiques sur F définissant l'ensemble des *états buts*.

Pour modéliser un tel problème, on peut utiliser un seul chronogramme tl avec $ats(tl) = \{s_f \mid f \in F\} \cup \{a\}$ (un attribut s_f pour chaque fluent $f \in F$, auquel on ajoute un attribut a permettant de représenter l'action courante), $do(ns(tl)) = [1..\infty]$ (au moins une étape correspondant à l'état initial), $\forall f \in F$, $ty(s_f) = state$, $do(s_f) = \{0, 1\}$ (attributs booléens de type $state$), $ty(a) = event$, $do(a) = A \cup \{\perp\}$ (attribut de type $event$) et les contraintes suivantes.

Spécification de l'état initial :

$$\begin{aligned} \forall f \in I\,, \quad & s_{f,1} = 1 \\ \forall f \in F \setminus I\,, \quad & s_{f,1} = 0 \end{aligned} \tag{1}$$

Pas d'action initiale et une action à chaque étape à partir de la deuxième :

$$\begin{aligned} & a_1 = \perp \\ \forall i \in [2..ns(tl)]\,, \quad & a_i \neq \perp \end{aligned} \tag{2}$$

Préconditions des actions :

$$\forall f \in F, \forall i \in [2..ns(tl)]\,,\ (f \in p_{a_i}) \rightarrow (s_{f,i-1} = 1) \tag{3}$$

Effets positifs, négatifs et nuls des actions :

$$\begin{aligned} \forall f \in F, \forall i \in [2..ns(tl)]\,, \quad & (f \in e^+_{a_i}) \rightarrow (s_{f,i} = 1) \\ & (f \in e^-_{a_i}) \rightarrow (s_{f,i} = 0) \\ & (f \in F \setminus (e^-_{a_i} \cup e^+_{a_i})) \rightarrow (s_{f,i} = s_{f,i-1}) \end{aligned} \tag{4}$$

Satisfaction des conditions but, avec $S_i = \{s_{f,i} \mid f \in F\}$:

$$\forall g \in G\,,\ g(S_{ns(tl)}) = true \tag{5}$$

On notera que les contraintes 1 et 2 (première ligne) sont obligatoires tandis que les contraintes de 2 (seconde ligne) à 5 sont optionnelles. On notera aussi la sémantique de la contrainte 5 ou apparaît la variable de dimension $ns(tl)$: à chaque valeur ns de $ns(tl)$, est associée la contrainte suivante : $\forall g \in G\,,\ g(S_{ns}) = true$.

3 Modélisation du problème de gestion de projets avec contraintes de ressources

Un RCPSP (Resource-Constrained Project Scheduling Problem (Baptiste *et al.*, 2001)) est défini par un n-uplet $\langle n, m, D, Co, Ca, Pr, Tmax\rangle$ où :

1. n est le nombre de *tâches* ;
2. m est le nombre de *ressources* ;
3. D est un tableau qui associe une *durée* D_t à chaque tâche $t \in [1..n]$;
4. Co est un tableau qui associe une *consommation* de ressource $Co_{t,r}$ à chaque tâche $t \in [1..n]$ et à chaque ressource $r \in [1..m]$;
5. Ca est un tableau qui associe une *capacité* Ca_r à chaque ressource $r \in [1..m]$: à chaque instant, la somme des consommations de la ressource r par toutes les tâches actives doit être inférieure ou égale à Ca_r ;
6. Pr est un tableau qui associe à chaque paire de tâches $t, t' \in [1..n]$ un booléen $Pr_{t,t'}$ égal à 1 si et seulement si t doit *précéder* t' ;
7. $Tmax$ est la *durée maximale* du projet : toutes les tâches doivent être terminées avant $Tmax$ ou à $Tmax$ au plus tard.

Pour modéliser un RCPSP, on peut utiliser n chronogrammes : un chronogramme tl_t pour chaque tâche $t \in [1..n]$, avec deux attributs par chronogramme : un attribut ac_t représentant le fait que la tâche t est active ou non et un attribut ti_t représentant les dates de début et de fin de t, soit $\forall t \in [1..n]$, $ats(tl_t) = \{ac_t, ti_t\}$, $do(ns(tl_t)) = \{3\}$ (trois étapes par chronogramme correspondant à l'instant 0 et au début et à la fin de t), $ty(ac_t) = state$, $do(ac_t) = \{\top, \bot\}$ (attribut de type $state$, avec deux valeurs possibles correspondant respectivement à l'état actif et inactif), $ty(ti_t) = time$, $do(ti_t) = [0..Tmax]$ (attribut de type $time$ prenant sa valeur entre 0 et $Tmax$) et les contraintes suivantes.

Instant initial :

$$\forall t \in [1..n]\,, \quad ti_{t,1} = 0 \tag{6}$$

Activité de chaque tâche aux étapes 1, 2 et 3 (si une tâche t débute à l'instant 0 ($ti_{t,1} = ti_{t,2} = 0$), on a $ac_{t,1} = ac_{t,2} = \top$ du fait de l'hypothèse que, si deux étapes successives sont temporellement identiques, elles sont identiques sur l'ensemble des attributs) :

$$\begin{aligned} \forall t \in [1..n]\,, \quad & (ti_{t,2} > ti_{t,1}) \rightarrow (ac_{t,1} = \bot) \\ & ac_{t,2} = \top \\ & ac_{t,3} = \bot \end{aligned} \tag{7}$$

Durée de chaque tâche :

$$\forall t \in [1..n]\,, \quad ti_{t,3} - ti_{t,2} = D_t \tag{8}$$

Précédences entre tâches :

$$\forall t, t' \in [1..n] \mid (Pr_{t,t'} = 1)\,, \quad ti_{t,3} \leq ti_{t',2} \tag{9}$$

Respect de la capacité de chaque ressource (pour chaque ressource, au début de chaque tâche, la somme de consommations de toutes les tâches actives ne dépasse pas la capacité de la ressource) :

$$\forall r \in [1..m], \forall t \in [1..n]\,, \quad (\sum_{t'=1}^{n} (val(ac_{t'}, tl_t, 2) = \top) \cdot Co_{t',r}) \leq Ca_r \tag{10}$$

On notera que toutes les variables et contraintes sont obligatoires (variables de dimension fixées). Dans la contrainte 10, $val(ac_{t'}, tl_t, 2)$ est une expression, fonction des autres variables du problème, qui désigne la valeur de l'attribut $ac_{t'}$ à l'étape 2 du chronogramme tl_t, différent du chronogramme $tl_{t'}$ auquel appartient $ac_{t'}$. D'une façon générale, si at est un attribut de type $state$ appartenant au chronogramme $tl(at)$, si tl est un chronogramme différent de $tl(at)$, si tl et $tl(at)$ possèdent tous deux un attribut de type $time$ et si ces attributs sont comparables, soit $j = max(j' \in [1..ns(at)] \mid (ti(at))_{j'} \leq (ti(tl))_i$: j est la dernière étape de $tl(at)$ qui se situe avant $(ti(tl))_i$ ou juste à $(ti(tl))_i$; si j n'existe pas, $val(at, tl, i)$ n'est pas défini ; sinon, $val(at, tl, i) = at_j$, conformément à l'hypothèse qu'entre deux étapes successives, un attribut de type $state$ conserve la valeur qu'il a prise lors de la première étape (voir la figure 3). On fait aussi l'hypothèse classique dans le cadre CSP qu'une contrainte telle que $val(ac_{t'}, tl_t, 2) = \top$ équivaut à un booléen qui vaut 1 si la contrainte est satisfaite et 0 sinon.

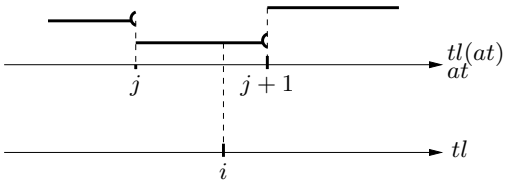


FIGURE 3 – Valeur d'un attribut at de type $state$ à une étape i d'un chronogramme tl, différent de $tl(at)$.

En fait, les attributs ac_t ne sont pas indispensables à la modélisation du RCPSP qui pourrait être modélisé en utilisant uniquement les attributs ti_t de type $time$. L'intérêt de leur introduction est de permettre une modélisation aisée d'extensions du RCPSP où, par exemple, plusieurs variantes avec plusieurs consommations et plusieurs durées existent pour chaque tâche.

4 Modélisation du problème de gestion de la mission d'une patrouille de drones

Le problème auquel nous nous intéressons ici est inspiré de la compétition internationale de micro-drones (Micro Air Vehicle Conference Competition, voir par exemple http ://www.mav07.org/). Ses données sont les suivantes :

1. un *temps* TI imparti à la mission ;
2. une *zone* HO de *décollage* et d'*atterrissage* des drones ;
3. un ensemble AI de *zones*, contenant chacune une cible *à identifier* ; pour identifier une cible, il est nécessaire d'avoir une caméra et d'effectuer un huit sur la zone associée ;
4. un ensemble AL de *zones*, contenant chacune une cible *à localiser* ; pour localiser une cible, il est nécessaire d'avoir une caméra et d'effectuer un balayage de la zone associée ;
5. un ensemble AT de *zones*, contenant chacune une cible *à toucher* ; pour toucher une cible, il est nécessaire d'avoir au moins une bille à lancer ;
6. un nombre NP de *drones* ;
7. un tableau CA indiquant pour chaque drone s'il dispose ou non d'une *caméra* ;
8. un tableau NM indiquant le nombre de *billes* dont dispose chaque drone ;
9. un tableau SP indiquant pour chaque drone sa *vitesse* de déplacement ;
10. un tableau DU indiquant pour chaque drone et chaque action parmi les actions $TOFF$, $LAND$, $EIGHT$, $SCAN$ et $DROP$ (décollage, atterrissage, huit, balayage et lancer) la *durée* estimée de cette action effectuée par ce drone ;
11. un tableau DI indiquant la *distance* à parcourir entre toute paire de zones, y compris la zone d'atterrissage et de décollage ;
12. une *durée minimale* $DUMIN$ entre deux décollages ou atterrissages de drones différents à HO.

On note $A = AI \cup AL \cup AT \cup \{HO\}$ l'ensemble des zones, supposées toutes distinctes. On note aussi $AC = \{TOFF, LAND, EIGHT, SCAN, DROP, GOTO, NOT\}$ (décollage, atterrissage, huit, balayage, lancer, déplacement et rien) l'ensemble des actions possibles.

Pour modéliser ce problème, on peut utiliser NP chronogrammes : un chronogramme tl_p pour chaque drone $p \in [1..NP]$, avec cinq attributs par chronogramme : un attribut fl_p représentant le fait que le drone p est *en vol* ou non, un attribut at_p représentant la *zone* où il se trouve, un attribut nm_p représentant le nombre de *billes* dont il dispose, un attribut ac_p représentant l'*action* qu'il effectue et un attribut ti_p représentant l'*instant* courant, soit $ats(tl_p) = \{fl_p, at_p, nm_p, ac_p, ti_p\}$, $do(ns(tl_p)) = [1..2 \cdot |A| + 3]$ (au moins une étape correspondant à l'instant 0 et au plus $2 \cdot |A| + 3$ étapes correspondant à la situation où toutes les cibles sont traitées par le même drone), $ty(fl_p) = state$, $do(fl_p) = \{\top, \bot\}$ (attribut de type $state$, avec deux valeurs possibles correspondant à en vol et au sol), $ty(at_p) = state$, $do(at_p) = A$ (attribut de type $state$, avec comme domaine l'ensemble A des zones), $ty(nm_p) = state$, $do(nm_p) = [0..NM_p]$ (attribut de type $state$, avec comme domaine l'ensemble des entiers entre 0 et NM_p), $ty(ac_p) = state$, $do(ac_p) = AC$ (attribut de type $state$, avec comme domaine l'ensemble AC des actions possibles) et $ty(ti_p) = time$, $do(ti_p) = [0, TI]$ (attribut de type $time$, avec comme domaine l'ensemble des instants entre 0 et TI). On a par ailleurs les contraintes suivantes.

État initial de chaque drone :

$$\forall p \in [1..NP], (fl_{p,1} = \bot) \wedge (at_{p,1} = HO) \wedge (nm_{p,1} = NM_p) \wedge (ti_{p,1} = 0) \tag{11}$$

État final de chaque drone :

$$\forall p \in [1..NP], (fl_{p,ns(tl_p)} = \bot) \wedge (at_{p,ns(tl_p)} = HO) \wedge (ac_{p,ns(tl_p)} = NOT) \tag{12}$$

Conditions et effets d'un décollage :

$$\begin{aligned} \forall p \in [1..NP], \forall i \in [1..ns(tl_p) - 1], \quad & (ac_{p,i} = TOFF) \rightarrow \\ & ((at_{p,i} = HO) \wedge (fl_{p,i} = \bot) \wedge \\ & (fl_{p,i+1} = \top) \wedge (ti_{p,i+1} - ti_{p,i} = DU_{p,TOFF})) \end{aligned} \tag{13}$$

Conditions et effets d'un atterrissage :

$$\forall p \in [1..NP], \forall i \in [1..ns(tl_p)-1]\,, \quad (ac_{p,i} = LAND) \rightarrow ((at_{p,i} = HO) \wedge (fl_{p,i} = \top) \wedge (fl_{p,i+1} = \bot) \wedge (ti_{p,i+1} - ti_{p,i} = DU_{p,LAND})) \tag{14}$$

Conditions et effets d'un huit :

$$\forall p \in [1..NP], \forall i \in [1..ns(tl_p)-1]\,, \quad (ac_{p,i} = EIGHT) \rightarrow ((fl_{p,i} = \top) \wedge (at_{p,i} \in AI) \wedge (CA_p = 1) \wedge (ti_{p,i+1} - ti_{p,i} = DU_{p,EIGHT}) \tag{15}$$

Conditions et effets d'un balayage :

$$\forall p \in [1..NP], \forall i \in [1..ns(tl_p)-1]\,, \quad (ac_{p,i} = SCAN) \rightarrow ((fl_{p,i} = \top) \wedge (at_{p,i} \in AL) \wedge (CA_p = 1) \wedge (ti_{p,i+1} - ti_{p,i} = DU_{p,SCAN}) \tag{16}$$

Conditions et effets d'un lancer :

$$\forall p \in [1..NP], \forall i \in [1..ns(tl_p)-1]\,, \quad (ac_{p,i} = DROP) \rightarrow ((fl_{p,i} = \top) \wedge (at_{p,i} \in AT) \wedge (nm_{p,i} - nm_{p,i+1} = 1) \wedge (ti_{p,i+1} - ti_{p,i} = DU_{p,DROP}) \tag{17}$$

Conditions et effets d'un déplacement :

$$\forall p \in [1..NP], \forall i \in [1..ns(tl_p)-1]\,, \quad (ac_{p,i} = GOTO) \rightarrow ((fl_{p,i} = \top) \wedge (at_{p,i+1} \neq at_{p,i}) \wedge (ti_{p,i+1} - ti_{p,i} = DI_{at_{p,i},at_{p,i+1}}/SP_p)) \tag{18}$$

Conditions d'une action nulle :

$$\forall p \in [1..NP], \forall i \in [1..ns(tl_p)]\,, \quad (ac_{p,i} = NOT) \rightarrow ((fl_{p,i} = \bot) \wedge (at_{p,i} = HO)) \tag{19}$$

Conditions de changement des attributs fl, at et nm :

$$\forall p \in [1..NP], \forall i \in [1..ns(tl_p)-1]\,, \quad (fl_{p,i+1} \neq fl_{p,i}) \rightarrow (ac_{p,i} \in \{TOFF, LAND\}) \quad (at_{p,i+1} \neq at_{p,i}) \rightarrow (ac_{p,i} = GOTO) \quad (nm_{p,i+1} \neq nm_{p,i}) \rightarrow (ac_{p,i} = DROP) \tag{20}$$

Chaque cible doit être traitée une fois et une seule :

$$\forall a \in AI\,, \quad \sum_{p \in [1..NP], i \in [1..ns(tl_p)]} ((at_{p,i} = a) \wedge (ac_{p,i} = EIGHT)) = 1 \tag{21}$$

$$\forall a \in AL\,, \quad \sum_{p \in [1..NP], i \in [1..ns(tl_p)]} ((at_{p,i} = a) \wedge (ac_{p,i} = SCAN)) = 1$$

$$\forall a \in AT\,, \quad \sum_{p \in [1..NP], i \in [1..ns(tl_p)]} ((at_{p,i} = a) \wedge (ac_{p,i} = DROP)) = 1$$

Distance minimum entre décollages et atterrissages :

$$\forall p, p' \in [1..NP] \,|\, p < p', \quad \forall i \in [1..ns(tl_p)], \forall i' \in [1..ns(tl_{p'})], ((ac_{p,i} \in \{TOFF, LAND\}) \wedge (ac_{p',i'} \in \{TOFF, LAND\})) \rightarrow ((ti_{p',i'} \geq ti_{p,i+1} + DUMIN) \vee (ti_{p,i} \geq ti_{p',i'+1} + DUMIN)) \tag{22}$$

Pas d'actions successives identiques et pas d'aterrissage suivi d'un décollage :

$$\forall p \in [1..NP], \forall i \in [1..ns(tl_p)-1]\,, \quad (ac_{p,i+1} \neq ac_{p,i}) \quad (ac_{p,i} = LAND) \rightarrow (ac_{p,i+1} \neq TOFF) \tag{23}$$

Actions nulles seulement en début et fin :

$$\forall p \in [1..NP], \forall i \in [2..ns(tl_p)-1]\,, \quad (ac_{p,i} \neq NOT) \tag{24}$$

On notera qu'une contrainte de ressource comme le fait de ne pas pouvoir enclencher une action de lancer sans avoir au moins une bille est imposée par le domaine des attributs nm_p et que le respect du temps imparti à la mission est imposé par le domaine des attributs ti_p.

5 Modélisation du problème de gestion d'un satellite de surveillance et d'observation de la Terre

Le problème que nous présentons maintenant est une version simplifiée du problème de gestion des observations et des téléchargements de données par un satellite autonome de surveillance et d'observation de la Terre. On peut trouver une description du problème complet, une modélisation de type chronogramme et des algorithmes de résolution approchée dans (Pralet & Verfaillie, 2008). Les données de la version simplifiée sont les suivantes :

1. un *horizon de planification* défini par une date de *début* STA et une date de *fin* END ;
2. des *caractéristiques du satellite* telles que la *mémoire* maximum disponible $MMmax$, les niveaux minimum et maximum d'*énergie* $ENmin$ et $ENmax$, la *puissance* délivrée par les panneaux solaires en période d'éclairement $Psun$, les *puissances* consommées par la plate-forme, par l'observation et par le téléchargement $Psat$, Pob et Pdl et la *vitesse de rotation* du miroir de visée MS ;
3. un *état initial* défini par des niveaux de *mémoire* et d'*énergie* disponibles MM_0 et EN_0 et une *orientation* du miroir de visée OR_0 ;
4. un *ensemble d'observations* OB à réaliser et à télédécharger avec, pour chaque observation $o \in OB$, la *mémoire* qu'elle requiert SZ_o, la *durée* de sa réalisation DO_o et celle de son téléchargement DDO_o, le nombre de *fenêtres de réalisation* possibles sur l'horizon de planification NO_o et, pour chaque fenêtre $k \in [1..NO_o]$, sa date de *début* $SO_{o,k}$ et l'*orientation* requise du miroir de visée $OR_{o,k}$;
5. un nombre de *fenêtres de téléchargement* sur l'horizon de planification ND avec, pour chaque fenêtre $k \in [1..ND]$, sa date de *début* SD_k et sa *durée* DD_k ;
6. un nombre de *fenêtres d'éclipse* du satellite sur l'horizon de planification NE avec, pour chaque fenêtre $k \in [1..NE]$, sa date de *début* SE_k et sa *durée* DE_k.

Pour modéliser ce problème, on utilise d'abord des variables CSP classiques qui peuvent être vues comme des chronogrammes à un seul attribut et une seule étape :

1. pour chaque observation $o \in OB$, sa *fenêtre de réalisation* $no_o \in [0..NO_o]$ et sa *fenêtre de téléchargement* $nd_o \in [0..ND]$ (les valeurs 0 représentant respectivement l'absence de réalisation et l'absence de téléchargement) ;
2. pour chaque observation $o \in OB$, ses dates de *début* et de *fin de réalisation* so_o et $eo_o \in [STA..END]$;
3. pour chaque fenêtre de téléchargement $k \in [1..ND]$, les dates de *début* et de *fin de téléchargement* dans cette fenêtre sd_k et $ed_k \in [SD_k..SD_k + DD_k]$.

On utilise ensuite un seul timeline tl qui représente l'évolution de l'état du satellite, avec six attributs : trois attributs ob, dl et ec représentant respectivement le fait que le satellite est *en observation* ou non, *en téléchargement* ou non et *en éclipse* ou non, deux attributs mm et en représentant respectivement la *mémoire* et l'*énergie* disponibles et un attribut ti représentant l'*instant* courant, soit $ats(tl) = \{ob, dl, ec, mm, en, ti\}$, $do(ns(tl)) = [2..NSmax]$, avec $NSmax = 2 \cdot (NO + ND + NE) + 2$ (au moins deux étapes correspondant au début et à la fin de l'horizon de planification et au plus $NSmax$ étapes correspondant à la situation où toutes les observations sont réalisées, toutes les fenêtres de téléchargement sont utilisées et toutes les dates de début et de fin de fenêtre sont différentes), $ty(ob) = ty(dl) = ty(ec) = state$, $do(ob) = do(dl) = do(ec) = \{0, 1\}$ (attributs de type $state$, avec deux valeurs possibles correspondant respectivement à en observation ou non, en téléchargement ou non et en éclipse ou non), $ty(mm) = ty(en) = state$, $do(mm) = [0..MMmax]$, $do(en) = [ENmin..ENmax]$ (attributs de type $state$) et $ty(ti) = time$, $do(ti) = [STA..END]$ (attribut de type $time$). On a alors les contraintes suivantes.

Dates de début et de fin de réalisation d'une observation : la date de début est la date de début de la fenêtre choisie ; la date de fin est la date de début plus la durée ; en cas de non réalisation, dates de début et de fin sont arbitrairement positionnées à STA :

$$\begin{aligned} \forall o \in OB\,, \quad & (no_o \neq 0) \rightarrow ((so_o = SO_{o,no_o}) \wedge (eo_o = so_o + DO_o)) \\ & (no_o = 0) \rightarrow (so_o = eo_o = STA) \end{aligned} \tag{25}$$

Dates de début et de fin d'un télédéchargement dans une fenêtre : la date de fin est la date de début plus la durée de tous les télédéchargements réalisés dans la fenêtre ; en cas d'absence de télédéchargement, dates de début et de fin sont arbitrairement positionnées au début de la fenêtre :

$$\forall k \in [1..ND]\,, \quad ed_k = sd_k + \sum_{o \in OB} (nd_o = k) \cdot DDO_o \tag{26}$$
$$(ed_k = sd_k) \rightarrow (ed_k = sd_k = SD_k)$$

Contraintes entre observation et télédéchargement : si une observation n'est pas réalisée, elle n'est pas télédéchargée ; si elle est réalisée, son télédéchargement débute après la fin de sa réalisation :

$$\forall o \in OB\,, \quad (no_o = 0) \rightarrow (nd_o = 0) \tag{27}$$
$$(no_o \neq 0) \rightarrow (eo_o \leq sd_{nd_o})$$

Contraintes entre observations : si deux fenêtres de réalisation de deux observations distinctes sont temporellement en conflit et si la première est choisie, la seconde ne peut pas l'être ; de plus, si une fenêtre de réalisation est en conflit avec la position initiale du miroir, elle ne peut pas être choisie :

$$\forall o, o' \in OB, \forall k \in [1..NO_o], \forall k' \in [1..NO_{o'}] \mid \tag{28}$$
$$((o \neq o') \wedge (SO_{o,k} \leq SO_{o',k'} < SO_{o,k} + DO_o + \frac{|OR_{o,k} - OR_{o',k'}|}{MS})) \;,\; (no_o = k) \rightarrow (no_{o'} \neq k')$$
$$\forall o \in OB, \forall k \in [1..NO_o] \mid SO_{o,k} < STA + \frac{|OR_{o,k} - OR_0|}{MS} \;,\; no_o \neq k \tag{29}$$

État initial du satellite :

$$(mm_1 = MM_0) \wedge (en_1 = EN_0) \wedge (ti_1 = STA) \tag{30}$$

Etats en observation, en téléchargement et en éclipse :

$$\forall i \in [1..ns(tl)]\,, \quad (ob_i = 1) \leftrightarrow (\vee_{o \in OB}(so_o \leq ti_i < eo_o)) \tag{31}$$
$$(dl_i = 1) \leftrightarrow (\vee_{k \in [1..ND]}(sd_k \leq ti_i < ed_k))$$
$$(ec_i = 1) \leftrightarrow (\vee_{k \in [1..NE]}(SE_k \leq ti_i < SE_k + DE_k))$$

Évolution de la mémoire : la mémoire disponible est égale à la mémoire précédemment disponible, plus ce qui est produit par les télédéchargements qui se terminent et moins ce qui est consommé par les observations qui débutent :

$$\forall i \in [1..ns(tl) - 1]\,, mm_{i+1} = mm_i + \sum_{o \in OB} ((ti_{i+1} = ed_{nd_o}) - (ti_{i+1} = so_o)) \cdot SZ_o \tag{32}$$

Évolution de l'énergie : l'énergie disponible est égale à l'énergie précédemment disponible, plus ce qui a été produit par les panneaux solaires si le satellite n'était pas en eclipse et moins ce qui a été consommé par la plate-forme, par l'observation si le satellite était en observation et par le télédéchargement s'il était en télédéchargement, avec un maximum égal à $ENmax$:

$$\forall i \in [1..ns(tl) - 1]\,, \tag{33}$$
$$en_{i+1} = min(ENmax, en_i + (ti_{i+1} - ti_i) \cdot ((1 - ec_i) \cdot Psun - ob_i \cdot P_{ob} - dl_i \cdot P_{dl} - Psat))$$

Instants successifs : soient TO, TD et TE respectivement les instants de début et de fin effectifs d'observation, de télédéchargement et d'éclipse ; on a : $TO = \cup_{o \in OB \mid so_o \neq eo_o}\{so_o, eo_o\}$, $TD = \cup_{k \in [1..ND] \mid sd_k \neq ed_k}\{sd_k, ed_k\}$ et $TE = \cup_{k \in [1..NE]}\{SE_k, SE_k + DE_k\}$; soit $T = TO \cup TD \cup TE \cup \{STA, END\}$; soit ST l'ensemble T trié par ordre croissant ; le nombre d'étapes est égal à $|T|$ et, à chaque étape i, l'instant courant est égal à ST_i :

$$union(TO, TD, TE, \{STA, END\}, T)$$
$$ns(tl) = |T| \tag{34}$$
$$sort(T, ST)$$
$$\forall i \in [1..ns(tl)]\;,\; ti_i = ST_i$$

$union$ est supposée être une contrainte globale qui impose que T soit l'union ensembliste (sans duplication) de TO, TD, TE et $\{STA, END\}$ et $sort$ une autre contrainte globale qui impose que ST soit une permutation de T ordonnée de façon croissante (voir le *Global Constraint Catalog*, http ://www.emn.fr/x-info/sdemasse/gccat/). On notera que les limitations en mémoire et en énergie sont imposées via les domaines des attributs mm et en.

6 Leçons et pistes de travail

Le cadre CNT est un cadre simple et basique : essentiellement des ensembles de variables de dimension variable et des contraintes sur ces variables. La première leçon de ces quelques exercices de modélisation est que ce caractère simple et basique est sa principale force par la possibilité qu'il offre de modéliser finement des phénomènes dynamiques complexes, tels que des évolutions concurrentes interdépendantes. La seconde leçon est que la tâche de modélisation est encore et toujours une activité humaine complexe où le modélisateur se retrouve face à de nombreuses alternatives qu'il lui est parfois difficile d'évaluer et où les risques d'oubli ou d'erreur ne sont pas négligeables.

Notre travail actuel porte essentiellement sur l'algorithmique à développer au dessus du cadre CNT, en particulier dans le cas où le domaine de certaines variables de dimension n'est pas borné. Mais nous avons d'ores et déjà pu modéliser et résoudre les deux exemples présentés, où le nombre d'étapes est borné, grâce à des outils standard de Programmation par Contraintes, tels que*OPL Studio* (http ://www.ilog.com/products/-oplstudio/ Hentenryck (1999)) et *Comet* (http ://www.comet-online.org/ Hentenryck & Michel (2005) ; voir par exemple (Pralet & Verfaillie, 2008)) : la possibilité d'étapes nulles permet de gérer le nombre variable d'étapes.

Des travaux futurs devraient permettre d'étendre le cadre CNT à des évolutions des variables d'état plus complexes que de simples évolutions par paliers, à l'optimisation via des fonctions locales d'utilité (contraintes souples Schiex *et al.* (1995); Bistarelli *et al.* (1999)) et à la décision séquentielle dans l'incertain via des fonctions locales de plausibilité (contraintes souples d'un autre type Pralet *et al.* (2007)).

Références

BAPTISTE P., PAPE C. L. & NUIJTEN W. (2001). *Constraint-based Scheduling : Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers.

BISTARELLI S., MONTANARI U., ROSSI F., SCHIEX T., VERFAILLIE G. & FARGIER H. (1999). Semiring-Based CSPs and Valued CSPs : Frameworks, Properties and Comparison. *Constraints*, **4**(3), 199–240.

FIKES R. & NILSSON N. (1971). STRIPS : a New Approach to the Application of Theorem Proving. *Artificial Intelligence*, **2**, 189–208.

GHALLAB M., NAU D. & TRAVERSO P. (2004). *Automated Planning : Theory and Practice*. Morgan Kaufmann.

HENTENRYCK P. V. (1999). *The OPL Optimization Programming Language*. MIT Press.

HENTENRYCK P. V. & MICHEL L. (2005). *Constraint-based Local Search*. MIT Press.

MITTAL S. & FALKENHAINER B. (1990). Dynamic Constraint Satisfaction Problems. In *Proc. of the 8th National Conference on Artificial Intelligence (AAAI-90)*, p. 25–32, Boston, MA, USA.

PRALET C. & VERFAILLIE G. (2008). Decision upon Observations and Data Downloads by an Autonomous Earth Surveillance Satellite. In *Proc. of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-08)*, Los Angeles, CA, USA.

PRALET C., VERFAILLIE G. & SCHIEX T. (2007). An Algebraic Graphical Model for Decision with Uncertainties, Feasibilities, and Utilities. *Journal of Artificial Intelligence Research*, **29**, 421–489.

R. ROSSI, P. V. BEEK & T. WALSH, Eds. (2006). *Handbook of Constraint Programming*. Elsevier.

SCHIEX T., FARGIER H. & VERFAILLIE G. (1995). Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, p. 631–637, Montréal, Canada.

VERFAILLIE G., PRALET C. & LEMAÎTRE M. (2008). Constraint-based Modeling of Discrete Event Dynamic Systems. *Journal of Intelligent Manufacturing, Special Issue on "Planning, Scheduling, and Constraint Satisfaction"*. To appear.

Vers un modèle de planification hybride pour la planification d'action et de déplacement

Julien Guitton, Jean-Loup Farges

ONERA, DCSD - Centre de Toulouse
Unité de recherche Conduite et Décision
2 avenue Edouard Belin, 31055 Toulouse
{julien.guitton,jean-loup.farges}@onera.fr

Résumé : Les techniques actuelles de planification permettent de résoudre des problèmes académiques de plus en plus complexes. Cependant les problèmes du monde réel comme par exemple les problèmes pour lesquels un robot doit agir et se déplacer dans l'environnement ne présentent toujours pas de méthodes génériques de résolution satisfaisantes. Dans cet article, nous essayons de démontrer clairement les avantages de l'utilisation d'une architecture de planification hybride pour la résolution de tels problèmes. Cette architecture est composée d'un planificateur de tâches et d'un planificateur spécialisé dédié aux calculs des chemins. Nous mettons en valeur les gains de performance relatifs à un calcul de chemin par une méthode adéquate sur un benchmark couramment utilisé en planification : le domaine des rovers. Puis nous proposons et étudions différentes méthodes pour coupler un planificateur hiérarchique (HTN) avec un planificateur de chemins. Cette étude montre qu'une approche dans laquelle la recherche d'un plan symbolique et la recherche d'un ensemble de chemins correspondants sont entrelacées produit de meilleurs performances que les approches hiérarchiques actuellement utilisées. Enfin, nous proposons de remplacer les opérateurs de navigation du domaine de planification par un ensemble de préconditions géométriques. Ces préconditions sont ajoutées aux tâches nécessitant un déplacement pour être réalisées.

Mots-clés : planification de tâches, planification de chemin, architecture, robotique mobile

1 Introduction

Les techniques actuelles de planification permettent de résoudre des problèmes académiques de plus en plus complexes. Cependant, la gestion de problèmes plus réalistes reste peu abordée. En effet, les difficultés rencontrées lors de la résolution de tels problèmes proviennent de la nécessité d'avoir, d'une part, une base de connaissance spécifique et, d'autre part, des raisonnements particuliers tels que la recherche d'un chemin ou encore l'ordonnancement de l'utilisation de ressources.

Les planificateurs peuvent être classés en trois catégories (Ghallab *et al.*, 2004) : les planificateurs spécifiques au domaine, les planificateurs indépendants du domaine et les planificateurs dits configurables. Les premiers sont développés spécifiquement pour un problème ou type de problème donné. Ils résolvent ce problème rapidement et efficacement mais ne peuvent pas être utilisés dans des domaines différents. Les planificateurs indépendants du domaine, sont conçus pour résoudre une large classe de problèmes. Ils sont principalement utilisés pour la résolution de problèmes académiques. En effet, même s'il n'y a pas de restriction sur le type de problèmes qu'ils sont capables de résoudre, ils sont restreints à des domaines de planification classique, ie. des domaines satisfaisants un ensemble d'hypothèses de travail (Nau, 2007) : le planificateur doit avoir une connaissance complète du monde. Le système doit être déterministe, statique et fini. Les buts à réaliser doivent être explicites et le temps implicite. Même avec ces restrictions, pour des problèmes complexes, l'espace d'états à parcourir peut être trop grand pour que ceux-ci soient résolus en un temps de calcul réaliste. Les planificateurs configurables s'appuient sur les principes des planificateurs indépendants du domaine mais utilisent des connaissances spécifiques au domaine afin de restreindre la taille de l'espace de recherche. Cette connaissance peut être exprimée sous la forme d'un réseau de tâches hiérarchiques (Hierarchical task Network, HTN)(Erol *et al.*, 1994; Nau *et al.*, 2003) ou de règles de contrôle (Bacchus & Kabanza, 2000). Cette classe de planificateur permet de résoudre les problèmes plus efficacement, cependant seulement certains types de connaissances peuvent être représentés à l'aide de ces techniques (Wilkins & desJardins, 2001).

Dans cet article, nous nous focaliserons sur les problèmes pour lesquels un robot mobile doit agir et se déplacer dans l'environnement. La difficulté de cette classe de problèmes est due à la complexité inhérente aux calculs des déplacements du robot. En effet, la recherche d'un chemin permettant de réaliser les objectifs du robot n'est pas aisément gérée par un planificateur symbolique.

Long et Fox (Long & Fox, 2000) proposent une méthode pour identifier et exploiter automatiquement les connaissances particulières d'un domaine de planification telles que, par exemple, les données concernant les mouvements nécessaires d'un robot pour remplir la mission qui lui a été assignée. Ces connaissances extraites du domaine peuvent ensuite être utilisées par un planificateur spécialisé. L'idée d'utiliser des modules de raisonnement spécialisés pour aider un planificateur symbolique n'est pas nouvelle. Dans (Kambhampati *et al.*, 1993), les auteurs proposent un modèle de planification hybride qui utilise un ensemble de modules spécialisés pour augmenter le pouvoir d'expression et les capacité de raisonnement d'un planificateur classique. Cette approche est mise en œuvre dans (Lamare & Ghallab, 1998) pour la recherche d'une solution pour les problèmes dans lesquels un robot doit se déplacer, en couplant le planificateur IxTeT (Laborie & Ghallab, 1995) avec un planificateur d'itinéraire basé sur un graphe d'accessibilité. Dans (Cambon *et al.*, 2004), un planificateur intégrant un raisonnement symbolique et un raisonnement géométrique est utilisé pour la résolution des problèmes de saisie et positionnement d'un objet. Cependant ces travaux, d'une part, ne présentent pas l'intérêt d'une telle approche en termes de temps de calcul et de qualité de la solution et, d'autre part, n'expliquent pas comment connecter efficacement le planificateur principal avec le module de raisonnement spécialisé (Guitton *et al.*, 2008).

L'objectif de cet article est de démontrer que l'utilisation d'un planificateur spécialisé dédié aux sous-problèmes de recherche d'un chemin améliore le temps de calcul et la qualité du plan solution ainsi que de présenter une architecture de planification dans laquelle les deux planificateurs sont couplés de façon optimale. Dans la première partie, une comparaison est effectuée entre les résultats fournis par un planificateur symbolique et le modèle de planification hybride. Puis nous proposons et étudions différentes méthodes permettant de coupler les deux planificateurs. Enfin nous proposons une nouvelle vision de la spécification des déplacements dans laquelle ceux-ci ne sont plus considérés comme des actions à planifier mais comme des préconditions aux actions nécessitant un déplacement du robot pour être réalisées. Nous présentons également les résultats obtenus en dotant le planificateur spécialisé de la capacité de fournir des heuristiques permettant de guider la recherche du plan solution.

2 Vers un modèle de planification hybride

Les planificateurs indépendants du domaine développés actuellement sont de plus en plus performants et permettent de résoudre des problèmes de plus en plus complexes. Cependant, comme ils ne sont pas destinés à résoudre un type de problème particulier, leurs concepteurs tendent à les rendre le plus générique possible afin de pouvoir comparer leurs performances avec les planificateurs qui font autorité dans la communauté planification. Par conséquence, les problèmes utilisés (les benchmarks) sont souvent idéalisés ou sont des versions très simplifiées de problèmes réels (Howe & Dahlman, 2002).

La capacité de ces planificateurs à pouvoir résoudre un grand échantillon de problèmes différents a ses inconvénients : certains sous-problèmes pourrait être résolus de manière plus efficace par un module de raisonnement adapté au type de ces sous-problèmes. C'est le cas des sous-problèmes de planification de chemins dans les problèmes de planification pour un robot mobile.

Nous proposons de faire une étude de comparaison des performances entre une approche pour laquelle un planificateur symbolique doit résoudre seul la totalité du problème et une approche hybride pour laquelle les sous-problèmes de recherche d'un chemin sont délégués à un planificateur spécialisé. Dans les deux approches, un planificateur HTN similaire à SHOP2 (Nau *et al.*, 2003) est utilisé. La comparaison est effectuée sur le *domaine des rovers*.

2.1 Définition des problèmes-tests

Dans le domaine des rovers, utilisé lors de la troisième compétition de planification (Third International Planning Competition, 2002) et inspiré par les problèmes de rovers destinés à l'exploration de planètes, un ensemble de robots doivent collecter des échantillons de roches, prendre des photos et transmettre leurs résultats à une station fixe. Les robots doivent en outre se déplacer dans l'environnement pour remplir leur mission.

Le problème que nous utilisons pour comparer la méthode de planification classique avec la méthode hybride est le suivant : un rover unique doit collecter des échantillons à des points de l'environnement préalablement définis. Le rover doit se déplacer dans l'environnment tout en évitant les obstacles. L'environnement, représenté sur la figure 1, est composé d'obstacles (en gris) et de zones sur lesquelles les prélèvements d'échantillon doivent être faits (en jaune). Le point vert représente la position initiale du rover.

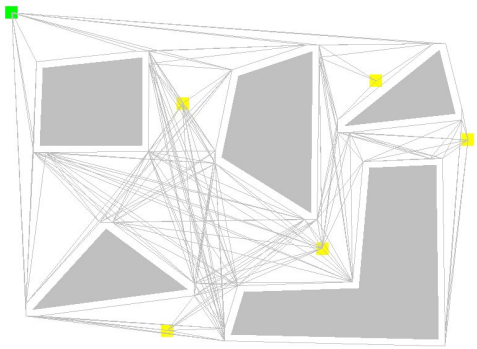


FIGURE 1 – la carte de l'environnement et le graphe de visibilité associé

Le domaine de planification et le problème sont formalisé avec un langage proche [1] du formalisme de planification PDDL (Fox & Long, 2003). L'état initial est composé de prédicats symboliques modélisant le rover et les ressources disponibles, ainsi que l'environnement sous la forme de prédicats représentant les points de passage et les chemins possibles entre ces points de passage.

Ces données géométriques sont extraites du graphe de visibilité [2] associé à la carte de l'environnement (figure 1). Les graphes de visibilité sont souvent utilisés lors de la modélisation de l'environnement dans les applications robotiques. Ils permettent de représenter les chemins les plus courts que peut emprunter un véhicule tout en évitant les obstacles. Dans un souci de réalisme, les obstacles ont été augmentés à l'aide d'une somme de Minkowski (Ramkumar, 1996), ainsi le robot doit respecter une distance de sécurité à proximité des obstacles.

Les sommets du graphe de visibilité sont transformés en points de passage symboliques et l'ensemble des arcs représentent les chemins possibles entre les points de passage. Ces chemins sont exprimés à l'aide du prédicat symbolique `can_traverse` comme dans la version originale du problème :

```
(waypoint waypoint0)(waypoint waypoint1)
(can_traverse rover1 waypoint0 waypoint1)
```

2.2 Méthode classique

Dans la méthode classique, le processus de planification est effectué entièrement par un planificateur hiérarchique (HTN).

En planification classique le but est, partant d'un état initial décrivant l'état du monde à l'instant initial, d'atteindre un état final dans lequel tous les objectifs sont réalisés. Cet état final est atteint en appliquant successivement des opérateurs de planification représentant les actions réalisables dans l'environnement. La sélection d'un opérateur à appliquer est effectuée en testant ses préconditions avec l'état courant du monde. Puis ses effets sont appliqués, correspondant ainsi à une transition entre deux états.

En planification HTN, l'objectif n'est pas de satisfaire un ensemble de buts mais de décomposer une série de tâches. Le domaine de planification contient un ensemble de méthodes spécifiant la manière de décomposer une tâche en sous-tâches et un ensemble d'opérateurs correspondant aux tâches primitives, ie. aux tâches non décomposables. Un problème de planification est décrit par un état initial et un ensemble de tâches de haut niveau à décomposer jusqu'à obtenir une séquence de tâches primitives correspondant aux actions que le robot peut réaliser dans l'environnement.

La recherche d'un chemin dans l'environnement permettant au rover d'accomplir sa mission est effectuée en unifiant les préconditions de l'opérateur `navigate` avec l'état courant.

1. nous utilisons un formalisme intégrant les notions de décomposition HTN
2. le graphede visibilité associé à la carte de l'environnement comporte 45 nœuds et 434 arêtes

```
(Operator (!navigate ?r ?x ?y)
 ;; preconditions
 ((rover ?r)(waypoint ?x)(waypoint ?y)
  (at ?r ?x)(can_traverse ?r ?x ?y))
 ;; effects
 ((not(at ?r ?x))(at ?r ?y))
)
```

L'opérateur `navigate` est appelé par une méthode dont le rôle est d'éviter au rover de tourner en rond dans l'environnement, en retenant les points de passage déjà visités.

2.3 Modèle de planification hybride

Dans la méthode de planification hybride, les sous-problèmes de planification d'un chemin sont délégués à un planificateur spécialisé au lieu d'être résolus par le planificateur de tâches. Ce planificateur de déplacement s'appuie sur un graphe de visibilité et l'algorithme de Dijkstra (Dijkstra, 1959) pour trouver le chemin le plus court entre deux points de passage spécifiés par le planificateur de tâches.

Le planificateur HTN et le planificateur spécialisé sont liés grâce à l'utilisation de préconditions géométriques au niveau des opérateurs de navigation. Lorsque le planificateur de tâches rencontre ce type de précondition, il envoie une requête au planificateur de déplacement. Cette requête contient le point initial et le point final du déplacement. S'il existe un chemin entre ces deux points, alors le planificateur spécialisé renvoie une réponse positive et le processus de planification continue. Dans le cas contraire, le planificateur HTN annule la dernière action planifiée et cherche une nouvelle solution.

```
(Operator (!navigate ?r ?x ?y)
 ;; preconditions
 ((rover ?r)(waypoint ?x)(waypoint ?y)
  (at ?r ?x)(posX ?x ?c1x)(posY ?x ?c1y)
  (posX ?y ?c2x)(posY ?y ?c2y))
 ;; geometrical data
 ((from ?c1x ?c1y)(to ?c2x ?c2y))
 ;; effects
 ((not(at ?r ?x))(at ?r ?y))
)
```

Dans cette version du problème, l'état initial ne contient pas les prédicats `can_traverse`. Cependant, nous avons ajouté des prédicats spécifiant les coordonnées des points de passage. Ce sont les données géométriques qui sont transmises au planificateur de déplacement lorsqu'une précondition géométrique est rencontrée :

```
(waypoint waypoint5)
(posX waypoint5 60)(posY waypoint5 99)
```

Cet ajout est effectué en utilisant les informations disponibles dans la carte de l'environnement avant le processus de planification. Comme nous utilisons un environnement statique et connu pour nos tests, le temps de génération de l'état initial n'est pas pris en compte dans cette approche ainsi que dans l'approche classique.

2.4 Comparaison entre la méthode classique et le modèle hybride

Afin de comparer la méthode classique et la méthode de planification hybride, nous avons exécuté chacune d'elles sur cinq problèmes de difficulté croissante. Dans le premier problème le rover doit prélever un seul échantillon. Dans les problèmes suivants, un nouvel objectif est ajouté à chaque fois. Tous les objectifs sont identiques, c'est-à-dire, collecter un échantillon sur une zone spécifiée.

Le temps de calcul nécessaire pour trouver un plan solution pour chaque méthode est représenté sur la figure 2. Le modèle hybride permet de trouver une solution en un temps inférieur à l'approche classique. De plus, cette différence de temps de calcul augmente fortement avec le nombre d'objectifs à atteindre. La

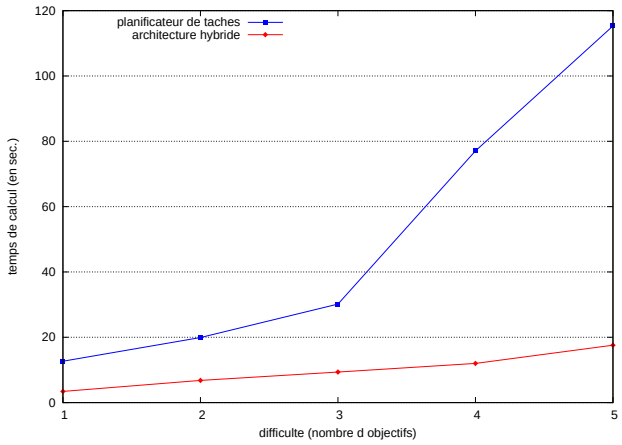


FIGURE 2 – Comparaison du temps de calcul d'un plan solution

principale raison est que, dans l'approche classique, la recherche d'un chemin n'est pas guidée. En effet, le planificateur de tâche cherche un chemin solution de la même façon que pour les tâches classiques, ie. en décomposant les tâches de haut niveau en sous-tâches jusqu'à obtenir des tâches élémentaires dont les préconditions peuvent être unifiées avec l'état courant du monde. Le planificateur ne tient pas compte des notions de distance, il choisit comme prochaine destination le point de passage représenté par le premier prédicat qu'il peut unifier avec l'état courant. Dans l'approche hybride, le planificateur de déplacement est spécialisé dans la gestion de ce type de sous-problèmes et peut donc fournir une solution en un temps minimal.

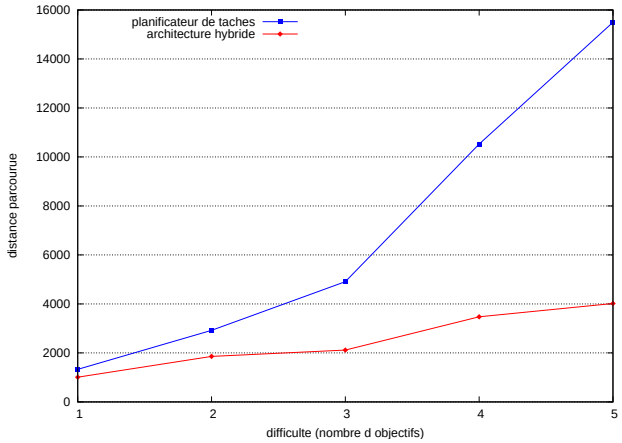


FIGURE 3 – Comparaison de la qualité de la solution en termes de distance parcourue

La comparaison de la distance totale parcourue par le rover pour réaliser les objectifs pour les deux méthodes est représentée sur la figure 3. La distance parcourue par le robot lorsque le modèle hybride est utilisé peut être utilisée pour juger la qualité de la solution fournie par l'approche classique. En effet, dans ces tests, l'ordre de réalisation des objectifs est préalablement fixé. Ainsi, comme le planificateur spécialisé s'appuie sur l'algorithme de Dijkstra, la distance totale est minimale. Ces résultats corroborent les résultats obtenus pour le temps de calcul : les sous-problèmes de recherche d'un chemin dans l'environnement pour un robot mobile ne sont pas facilement gérés par un planificateur de tâches.

Il est à noter que la recherche du plus court chemin est envisageable avec l'approche classique dans laquelle l'intégralité du problème est calculée par le planificateur de tâches. Nous avons encodé l'algorithme de Dijkstra sous la forme de méthodes et d'opérateurs de planification. La distance parcourue est identique à celle obtenue avec le planificateur spécialisé, cependant les temps de calcul d'un plan solution ont été multipliés par dix en moyenne. Ainsi, le plan pour le problème dans sa version la plus complexe (5 objectifs) a été calculé en plus de 1700 secondes.

Lors de la troisième compétition internationale de planification (IPC3), le planificateur SHOP2 faisait appel à des routines externes codées en Lisp pour calculer les meilleurs chemins sur les problèmes du domaine des rovers. Cette manière d'améliorer les performances du planificateur peut être vue comme l'utilisation d'un modèle hybride dans lequel certains sous-problèmes sont résolus par des algorithmes spécifiques.

3 Couplage des deux planificateurs

L'utilisation d'un planificateur spécialisé pour résoudre certains sous-problèmes bien identifiés permet de réduire le temps de calcul et d'améliorer la qualité de la solution. Cependant, la manière de coupler le planificateur spécialisé et le planificateur principal influe sur le temps de résolution du problème. Nous proposons d'étudier différentes méthodes de couplage entre un raisonnement symbolique et un raisonnement géométrique.

3.1 Différentes méthodes de couplage

Nous pouvons considérer plusieurs façon de lier le planificateur de tâches et le planificateur de déplacement. Une approche naïve consiste à définir une hiérarchie de planificateurs. Cette approche hiérarchique est l'approche classiquement utilisée lors de la conception d'une architecture robotique (Latombe, 1991). En effet, en règle générale, un plan symbolique est préalablement calculé. Puis ce plan est raffiné par un raisonneur spécialisé, ie. un chemin géométriquement faisable satisfaisant les actions symboliques *navigate* est recherché (Ghallab *et al.*, 2004). S'il n'existe pas de chemin satisfaisant ces contraintes, alors le planificateur de tâches calcule un nouveau plan symbolique.

Cette approche peut être améliorée en enregistrant les actions produisant des contraintes géométriques, ie. nécessitant un déplacement. Ainsi, lorsque le planificateur de déplacement ne trouve pas de solution permettant de raffiner le plan symbolique, le planificateur de tâches effectue un retour arrière à partir de l'action infaisable géométriquement.

Une autre solution pour coupler le planificateur principal avec le planificateur de déplacement est d'entrelacer leur exécution. Dans cette approche, les contraintes géométriques spécifiant un déplacement sont envoyées par le planificateur de tâches au planificateur spécialisé dès qu'elles sont rencontrées. Si le planificateur spécialisé trouve une solution permettant de déplacer le robot, alors il renvoie une réponse positive. Dans le cas contraire, ie. si les contraintes géométriques ne sont pas satisfiables, alors il renvoie un message d'erreur. Ces appels au planificateur spécialisé peuvent être vus comme des tests de satisfaction de préconditions.

3.2 Etude de performance des différentes méthodes

Ces trois approches (hiérarchique, hiérarchique avec backtrack et entrelacée) sont testées sur cinq exemples de difficulté croissante dans lesquels un robot doit prendre des photos de quatre objectifs. Le robot doit se déplacer dans l'environnement pour atteindre les objectifs. Pour chaque objectif, deux points d'action sont définis. Cependant, le premier de ces deux points, correspondant au premier point fourni par le planificateur de tâches, est possiblement inacessible (eg. le point est défini à l'intérieur d'un obstacle). Dans le premier exemple (Pb1-0), tous les points définis sont accessibles alors que, dans le second problème, le premier objectif n'est réalisable par le robot qu'en se positionnant aux coordonnées du point alternatif fourni. Dans les problèmes suivants, un objectif supplémentaire n'est pas accessible par le premier point. Dans le dernier problème, le robot doit se positionner au point alternatif de chaque objectif pour exécuter les actions afin de réaliser l'ensemble de la mission.

La figure 4 illustre le temps de calcul d'un plan solution pour les trois méthodes présentées permettant de coupler le planificateur de tâches avec le planificateur de déplacement sur les cinq problèmes. Comme nous l'avions prévu, utiliser les résultats du planificateur de déplacement pour chercher un nouveau plan à partir de l'action infaisable au niveau des déplacements offre de meilleurs résultats qu'une approche hiérarchique aveugle, ie. dans laquelle un nouveau plan est calculé à partir de l'état initial. Cependant la méthode entrelaçant l'exécution du planificateur de tâches et du planificateur de déplacement est la plus performante des trois méthodes. En effet, si le point sur lequel le robot doit se positionner n'est pas atteignable, alors un déplacement vers la position alternative peut immédiatement être requis par le planificateur de tâches.

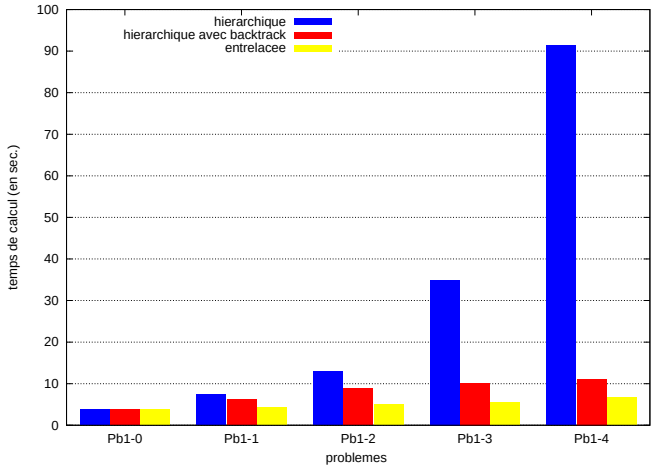


FIGURE 4 – Temps de calcul pour trouver un plan solution avec les trois approches

Problème	hiérarchique	hiérarchique avec backtrack	entrelacée
Pb1-0	4	4	4
Pb1-1	8	8	5
Pb1-2	17	11	6
Pb1-3	43	13	7
Pb1-4	112	14	8

TABLE 1 – Nombres de requêtes envoyées dans les trois approches

Le tableau 1 résume le nombre de requêtes envoyées par le planificateur de tâches au planificateur de déplacement. Ce nombre de requêtes est corrélé avev le temps de calcul. Cette corrélation entre le temps de calcul et le nombres de requêtes transmises indique un effet de causalité : la principale raison de l'augmentation du temps de calcul est une augmentation du nombre de chemins à calculer pour satisfaire l'ensemble de la mission.

Avec l'approche hybride entrelacée, le planificateur de tâches ne cherche pas à satisfaire le prochain objectif tant que les contraintes géométriques de l'objectif courant n'ont pas été satisfaites, ie. tant qu'un chemin permettant de satisfaire l'objectif courant n'a pas été calculé. Alors que, avec les deux méthodes hiérarchiques, un plan symbolique satisfaisant l'ensemble des objectifs est calculé avant de rechercher les chemins permettant réellement de les satisfaire. Ainsi, avec l'approche hiérarchique classique, les mêmes contraintes géométriques, possiblement non satisfiables, sont envoyées au planificateur de déplacement tant que le planificateur de tâches n'a pas atteint la définition du déplacement alternatif à l'action impossible lors de son processus de backtrack.

4 Améliorations de l'architecture hybride

Nous avons montré l'intérêt d'un modèle de planification couplant un planificateur de tâches avec un planificateur de déplacement pour les problèmes nécessitant la gestion des déplacements d'un robot mobile. Dans cette partie, nous proposons d'augmenter le langage de description du domaine de planification en se basant sur le postulat suivant : *les déplacements sont des préconditions aux actions*. Puis nous présentons une méthode permettant d'optimiser les solutions par l'utilisation d'heuristiques guidant la recherche du plan.

4.1 Vers une nouvelle description des opérateurs

Comme les chemins sont calculés par un planificateur spécialisé, la présence des opérateurs de navigation dans le domaine de planification n'est plus nécessaire. En effet, les déplacements du robot nécessaire à la

réalisation d'une action peuvent être vus comme étant des préconditions à cette action. Par exemple, pour effectuer un prélèvement le robot doit se déplacer jusqu'à la zone puis effectuer le prélèvement. Dans ce cas, la description de l'opérateur de planification correspondant à l'action de prélèvement d'un échantillon peut être augmentée avec les contraintes géométriques relatives à ce déplacement.

Ainsi nous proposons l'ajout de deux nouveaux types de préconditions appelées respectivement *préconditions d'attitude* et *préconditions de comportement*. Ces préconditions permettent d'exprimer un ensemble de contraintes géométriques nécessaires à la réalisation des actions. Les préconditions d'attitude permettent de définir l'attitude que doit avoir le robot en vue de la réalisation de l'action, eg. sa position, son orientation, etc. Les préconditions de comportement définissent le comportement du robot, en termes de déplacement, durant la réalisation de l'action. Ces dernières permettant de prendre en compte les déplacements dans le cas d'actions ayant une durée. À titre d'exemple, l'opérateur `film_objective` peut être décrit de la manière suivante :

```
(Operator (!film_objective ?r ?o)
  ;; symbolic preconditions
  ((rover ?r) (objective ?o)
   (camera ?c) (has_camera ?r ?c)
   (is_calibrated ?c))
  ;; attitude preconditions
  ((distance(r.pos, o.pos)>=10)
   (distance(r.pos, o.pos)<=20)
   (rel_angle(r.pos, o.pos, r.heading)=90)
   (r.speed = r.speed_max))
  ;; behaviour preconditions
  ((duration 10)
   (constant(r.heading))
   (constant(r.speed)))
  ;; effects
  ((has_film ?r ?o))
)
```

Dans cet exemple, la signification des préconditions d'attitude et de comportement est la suivante : le robot doit se positionner à une distance comprise entre 10 et 20 mètres perpendiculairement à l'objectif. Pendant l'action (qui a une durée de 10 secondes), il doit maintenir de façon constante son cap et sa vitesse.

Comme ces contraintes ne décrivent pas de façon unique un point de l'environnement, un échec lors de la satisfaction des préconditions de comportement ne signifie pas que l'action est irréalisable. La figure 5 représente les étapes de choix et d'application d'une action. Durant le processus de planification, le planificateur de tâches envoie au planificateur spécialisé une requête contenant les contraintes géométriques spécifiées par les préconditions d'attitude. S'il existe une solution permettant de satisfaire ces préconditions et que l'action nécessite un déplacement (ou plus généralement un comportement particulier) durant sa réalisation, alors le planificateur de tâches envoie une nouvelles requête au planificateur spécialisé. Cette seconde requête contient les contraintes géométriques relatives aux préconditions de comportement. Si toutes ces préconditions sont satisfaites, alors les effets de l'action peuvent être appliqués à l'état courant du monde. Dans le cas contraire, une autre action est testée. Si les préconditions de comportement ne sont pas applicables, alors le planificateur de déplacement recherche une nouvelle solution satisfaisant les préconditions d'attitude. Puis il tente à nouveau de satisfaire les conditions de comportement à partir de cette nouvelle position.

Ce nouveau formalisme d'expression des opérateurs de planification est principalement destiné à la méthode hybride entrelacée. Cependant, il peut être utilisé avec les approches hiérarchiques. Dans ce cas, les requêtes sont retenues en mémoire puis transmises au planificateur de déplacement dès qu'un plan symbolique a été trouvé.

4.2 Guidage du plan par heuristique

Bien que la méthode de couplage entracée soit plus performante que les méthodes hiérarchiques en termes de temps de calcul, la solution n'est pas globalement optimale pour les problèmes où l'ordre de réalisation des objectifs n'est pas préalablement fixé. En effet, le déplacement du robot est calculé par segments, ie.

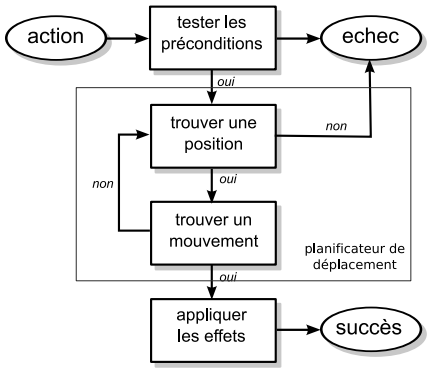


FIGURE 5 – Les différentes étapes de la sélection d'une action

entre deux zones d'action. Ainsi, nous proposons l'utilisation d'heuristiques calculées par le planificateur spécialisé permettant de guider la recherche du plan symbolique. Celui-ci envoie une *requête-conseil*. Cette requête-conseil ne modifie pas la position du robot mais déclenche des estimations au niveau du planificateur de déplacement dont les résultats permettent ensuite de guider le planificateur de tâches lors de la sélection d'une action. Par exemple, dans la description suivante, l'heuristique `distance_from_waypoint` est utilisée et permet de choisir comme prochain objectif, l'objectif le plus proche de la position actuelle du robot.

```
(Method (choose-objective)
 ;; choix d'un objectif
 (:sort-by ?distance < (
   (goal(has_soil_data ?wp))(at ?r ?x)
   (external(distance_from_waypoint ?r ?x ?wp ?distance)) ))
 ((realize-objective ?wp)(choose-objective))
)
```

L'utilisation de telles heuristiques permet d'optimiser les déplacements du robot. Le gain obtenu en termes de distance totale parcourue en utilisant comme heuristique pour le choix du prochain objectif à traiter, l'objectif le plus proche de la position actuelle du robot, est présenté sur la figure 6. Les problèmes utilisés sont ceux de la comparaison entre modèle classique et modèle hybride. L'ordre de réalisation des objectifs n'est pas fixé a priori.

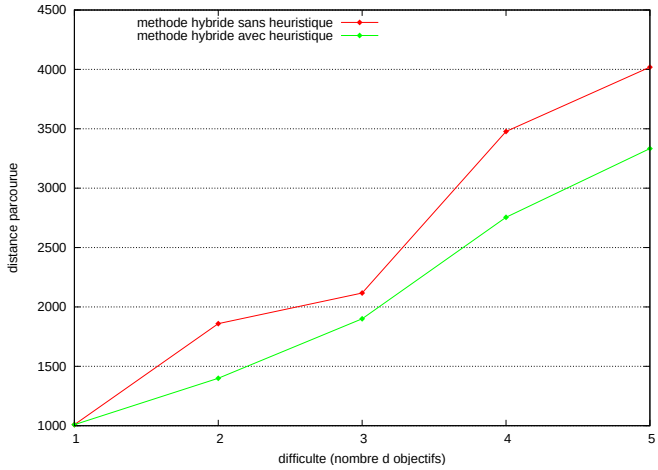


FIGURE 6 – Distance parcourue avec et sans heuristique

5 Conclusion

Dans cet article, nous avons essayé de démontrer les avantages à utiliser une architecture de planification hybride pour la résolution des problèmes nécessitant un raisonnement géométrique en plus du raisonnement symbolique classique. Dans cette architecture, les sous-problèmes de déplacement d'un robot mobile sont délégués par le planificateur de tâches à un planificateur spécialisé.

Les tests réalisés montrent clairement l'intérêt d'un tel modèle hybride de planification aussi bien en termes de temps de calcul qu'en termes de distance parcourue par le robot. Puis, les résultats obtenus sur les différentes approches permettant de coupler le planificateur de tâches avec le planificateur de déplacement montrent qu'une méthode dans laquelle l'exécution des deux planificateurs est entracée est plus performante qu'une méthode hiérarchique dans laquelle un plan symbolique est obtenu puis raffiné au niveau des déplacements. Le fonctionnement de l'architecture est le suivant : le planificateur de tâches fait appel au planificateur spécialisé en lui transmettant un ensemble de contraintes géométriques à satisfaire pour réaliser l'action courante. Ces contraintes sont exprimées au niveau des opérateurs sous la forme de préconditions d'attitude et de préconditions de comportement. Cependant, le principal inconvénient de cette architecture est la non optimalité globale de la solution en termes de déplacement. En effet, comme les requêtes sont envoyés par le planificateur de tâches à chaque fois que des contraintes géométriques sont rencontrées, les chemins permettant au robot de se déplacer dans l'environnment sont calculés localement puis aggrégés à la fin du processus de planification. Pour remédier à ce problème, nous proposons de guider la recherche du plan par des heuristiques fournis pas le planificateur de déplacement.

Dans la suite de nos travaux, nous envisageons d'aborder la gestion des variables partagées entre les deux planificateurs (eg. carburant) et la gestion des contraintes temporelles. Nous étudierons également la prise en compte des échecs du planificateur spécialisé pour proposer des alternatives au niveau du planificateur principal. Enfin, nous souhaitons étendre notre architecture de planification hybride aux problèmes multi-agents tels que la gestion d'une patrouille de drones aériens.

Références

BACCHUS F. & KABANZA F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, **116**, 123–191.

CAMBON S., GRAVOT F. & ALAMI R. (2004). aSyMov : Towards more realistic robot plans. In *International Conference on Automated Planning and Scheduling*.

DIJKSTRA E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, **1**, 269–271.

EROL K., HENDLER J. & NAU D. (1994). UMCP : A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, p. 249–254.

FOX M. & LONG D. (2003). PDDL 2.1 : an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, **20**, 61–124.

GHALLAB M., NAU D. & TRAVERSO P. (2004). *Automated planning : Theory and Practice*. Morgan Kaufmann.

GUITTON J., FARGES J.-L. & CHATILA R. (2008). A planning architecture for mobile robotics. In *1st Mediterranean Conference on Intelligent Systems and Automation*.

HOWE A. & DAHLMAN E. (2002). A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research*, **17**, 1–33.

KAMBHAMPATI S., CUTKOSKY M., TENENBAUM J. & LEE S. (1993). Integrating general purpose planners and specialized reasoners :case study of a hybrid planning architecture. In *IEEE transactions on Systems, Man and Cybernetics*, p. 1503–1518.

LABORIE P. & GHALLAB M. (1995). IxTeT : an integrated approach for plan generation and scheduling. In *INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, volume 1, p. 485–495.

LAMARE B. & GHALLAB M. (1998). Integrating a temporal planner with a path planner for a mobile robot. In *AIPS Workshop Integrating planning, scheduling and execution*, p. 144–151.

LATOMBE J. (1991). *Robot Motion Planning*. Kluwer Academic Publishers.

LONG D. & FOX M. (2000). Extracting route-planning : first steps in automatic problem decomposition. In *AIPS-00 Workshop on Analysing and Exploiting Domain Knowledge for Efficient Planning*.

NAU D. (2007). Current trends in automated planning. *AI Magazine*, **4**(4), 43–58.

NAU D., AU T.-C., ILGHAMI O., KUTER U., MURDOCK J. W., WU D. & YAMAN F. (2003). SHOP2 : An HTN planning system. *Artificial Intelligence Research*, **20**, 380–404.

RAMKUMAR G. (1996). An algorithm to compute the minkowski sum outer-face of two simple polygons. In *twelfth annual symposium on Computational geometry*, p. 234–241.

THIRD INTERNATIONAL PLANNING COMPETITION (2002). Hosted at the Artificial Intelligence Planning and Scheduling (AIPS) conference. http ://planning.cis.strath.ac.uk/ competition/.

WILKINS D. & DESJARDINS M. (2001). A call for knowledge-based planning. *AI Magazine*, **22**(1), 99–115.

Introduction de connaissances expertes en Bandit-Based Monte-Carlo Planning avec application au Computer-Go

Louis Chatriot, Sylvain Gelly, Jean-Baptiste Hoock, Julien Perez, Arpad Rimmel, Olivier Teytaud

TAO (Inria), LRI, UMR 8623 (CNRS - Univ. Paris-Sud),
bat 490 Univ. Paris-Sud 91405 Orsay, France, teytaud@lri.fr

Résumé : Nous présentons notre travail sur l'introduction de l'expertise go dans l'algorithme de BBMCP appliqué au jeu de go. Notre contribution inclue (i) les livres d'ouvertures (ii) le biais dans l'arbre des possibles (iii) l'amélioration de la partie Monte-Carlo et la mise en évidence des facteurs critiques de succès tels que le risque de perte de diversité dans le jeu aléatoire, l'utilité des contre exemples pour la conception de Monte-Carlo ou pour l'introduction du biais. Le programme résultant de cette étude a récemment obtenu une victoire en condition de temps standard face à un joueur professionnel de Go en 9x9.

1 Introduction

Les diverses définitions des termes du jeu de go utilisées dans cet article peuvent être trouvées sur le site internet `http://senseis.xmp.net`

Le Bandit Based Monte Carlo Planning, BBMCP Kocsis & Szepesvari (2006), est un outil récent pour les tâches de planification difficiles. D'impressionnants résultats ont déjà été publiés Coulom (2006); Gelly & Silver (2007). Dans ce papier, nous présentons l'introduction de connaissance d'experts pour un problème spécifique de prise de décision, le jeu de Go, un jeu séquentiel à somme nulle. Après avoir présenté les principes de l'algorithme BBMCP, nous présenterons notre travail sur les trois grandes parties de l'algorithme : (1) les livres d'ouverture (2) la partie bandit (les simulations dans l'arbre) (3) l'estimation de Monte-Carlo. Toutes les modifications sont développées dans MoGo qui gagna en 2007 la première partie en Blitz en 9x9 contre un joueur professionnel et en 2008 la première partie en temps standard (30 minutes par coté) contre un joueur professionnel.

L'algorithme BBMCP consiste à construire un arbre dans lequel chaque noeud est une situation de l'environnement considéré et les branches sont des actions pouvant être prises par l'agent. Le point important du BBMCP est que l'arbre est fortement déséquilibré : un biais est appliqué en faveur des parties importantes de l'arbre. Ainsi, l'intérêt est mis aux zones de l'arbre ayant une plus forte espérance de gain. Plusieurs algorithmes ont été proposés : UCTKocsis & Szepesvari (2006) (Upper Confidence Trees, i.e. UCB applied to Trees), qui porte son attention sur les noeuds de l'arbre ayant le plus grand nombre de simulations gagnantes *plus* un terme favorisant les noeuds peu explorés, AMAF Gelly & Silver (2007) (All Moves As First), qui fait un compromis entre le score à la UCB1 et des informations heuristiques obtenues par permutations de simulations, et BAST Coquelin & Munos (2007) (Bandit Algorithm for Search in Trees), qui se focalise quant à lui sur le nombre total de noeuds dans l'arbre. D'autres algorithmes proches sont également disponibles dans Chaslot *et al.* (2007).

Dans le contexte du jeu de Go, les noeuds de l'arbre incorporent la configuration du tableau de jeu qu'ils représentent (aussi appelée coloration du plateau), et des statistiques telles que le nombre de victoires obtenues et le nombre total de parties jouées traversant ce noeud. Comme expliqué dans l'Algorithme 1, BBMCP utilise ces statistiques afin d'étendre itérativement l'arbre des possibles dans la région où la plus forte espérance de gain aura été calculée. Après chaque simulation à partir de la racine de l'arbre, les statistiques de gain et de perte sont mises à jour dans tous les noeuds ayant été parcourus et un nouveau noeud est créé dans l'arbre ; ce nouveau noeud correspond au premier noeud visité lors de la partie simulation Monte-Carlo (i.e. la première situation rencontrée dans la simulation et qui n'était pas encore dans l'arbre).

Ainsi, afin de concevoir des simulations à partir de la racine de l'arbre (la position courante) jusqu'à la position finale (victoire ou défaite), plusieurs décisions doivent être prises : (i) dans l'arbre, jusqu'à

parvenir à une feuille (ii) hors de l'arbre, jusqu'à ce que la partie soit terminée. Les buts de ces deux choix de transition sont différents. Dans l'arbre, la transition doit assurer un bon compromis entre exploration et exploitation. Hors de l'arbre, les simulations, habituellement nommées simulation Monte-Carlo, doivent permettre d'obtenir une bonne estimation de la probabilité de victoire.

L'Algorithme 1 résume le planning par méthode Monte-Carlo ; une version plus détaillée est donnée en Algorithme 4.

Algorithm 1 Pseudo-code d'un algorithme BBMCP appliqué à un jeu à somme nulle (comme le Go ou les échecs). T est un arbre de positions, avec chaque noeud contenant des statistiques (nombre de parties gagnées et perdues dans les simulations passant par ce noeud). En ce qui concerne la décision à la toute fin du pseudo-code, choisir le coup le plus simulé est la solution la plus sure ; d'autres solutions comme le choix du meilleur "ratio nombre de parties gagnées sur nombre de parties total" ne sont pas robustes car il peut y avoir très peu de simulations. Ici la récompense pour chaque partie est binaire (gagnée ou perdue) mais des distributions arbitraires peuvent être utilisées.

Initialiser T à un noeud unique, représentant l'état courant.
while Temps restant > 0 **do**
  Simuler une partie jusqu'à une feuille (une position) L de T (à l'aide de l'algorithme de bandit).
  Choisir un fils (un successeur) L' de L.
  Simuler une partie depuis la position L' jusqu'à la fin de la partie.
  Développer l'arbre : ajouter L' en tant que fils de L dans T.
  Mettre à jour les statistiques dans tout l'arbre. Dans UCT, chaque noeud connaît combien de simulations gagnées (depuis ce noeud) ont été jouées ainsi que le nombre de simulations totales. Pour d'autres formes d'exploration d'arbre, il peut y avoir besoin de plus d'informations (information heuristique dans le cas d'AMAF, nombre total de noeuds dans le cas de BAST Coquelin & Munos (2007)).
end while
Retourner le coup qui a été simulé le plus souvent à partir de la racine.

La fonction utilisée pour prendre les décisions hors de l'arbre est définie dans l'algorithme 2. Un atari se produit lorsqu'une chaîne (un groupe de pierres) peut être capturé en un coup. Des connaissances de Go ont été ajoutées dans cette partie, sous la forme de patterns 3x3 conçus pour jouer des parties plus réalistes.

Algorithm 2 Algorithme permettant de choisir un coup dans une simulation MC dans le cas du jeu de Go.

if le dernier coup est un atari **then**
  Sauver les pierres en atari.
else
  if il y a un emplacement vide parmi les 8 emplacements autour du dernier coup qui correspond à un pattern **then**
    Jouer au hasard sur l'un de ces emplacements (sortir).
  else
    if il y a un coup qui capture des pierres **then**
      Capturer les pierres (sortir).
    else
      if il reste des coups légaux **then**
        Jouer au hasard un coup légal (sortir).
      else
        Renvoyer "passe".
      end if
    end if
  end if
end if

La fonction utilisée pour prendre les décisions dans l'arbre est présentée dans l'algorithme 3. Cette fonction est l'élément principal de notre programme : elle décide de la direction dans laquelle l'arbre sera étendu. Il existe plusieurs formules différentes ; toutes sont basées sur l'idée de compromis entre exploitation (simuler dans le sens des coups à succès) et l'exploration (simuler les coups qui ne l'ont pas encore été ou peu).

Algorithm 3 Algorithme permettant de choisir un coup dans l'arbre, pour un jeu à somme nulle avec une récompense binaire (l'extension pour le cas de distributions arbitraires est directe). $Sims(s, d)$ est le nombre de simulations commençant à s avec comme premier coup d. Le nombre total de simulations pour une situation s est $Sims(s) = \sum_d Sims(s, d)$. Nous présentons ici différentes formules pour calculer le score (voir Lai & Robbins (1985); Auer *et al.* (2002); Gelly & Silver (2007) pour UCB1, UCB-Tuned et AMAF respectivement) ; d'autres variantes très importantes (pour le cas de domaines infinis, d'un très grand nombre de bras, de suppositions spécifiques) peuvent être trouvées dans Banks & Sundaram (1992); Agrawal (1995); Dani & Hayes (2006); Coquelin & Munos (2007); Chaslot *et al.* (2007).

Fonction $decision = Bandit(situation\ s\ in\ the\ tree)$.
for d dans un ensemble de décisions possibles **do**
  Soit $\hat{p}(d) = Wins(s, d)/Sims(s, d)$.
  switch (formule de bandit) :
- **UCB1** : calculer $score(d) = \hat{p}(d) + \sqrt{2\log(Sims(s))/Sims(s, d)}$.
- **UCB-Tuned.1** : calculer $score(d) = \hat{p}(d) + \sqrt{\hat{V}\log(Sims(s))/Sims(s, d)}$ avec $\hat{V} = \max(0.001, \hat{p}(d)(1 - \hat{p}(d)))$.
- **UCB-Tuned.2** : calculer $score(d) = \hat{p}(d) + \sqrt{\hat{V}\log(Sims(s))/Sims(s, d)} + \log(Sims(s))/Sims(s, d)$ avec $\hat{V} = \max(0.001, \hat{p}(d)(1 - \hat{p}(d)))$.
- **exploration guidée par AMAF** : Calculer $score(d) = \alpha(d)\hat{p}(d) + (1 - \alpha(d))\hat{\hat{p}}(d)$ avec :
  - $\hat{\hat{p}}(d)$ la proportion de simulations gagnées parmis les simulations AMAF utilisant la décision d dans la situation s ;
  - $\alpha(d)$ un coefficient dépendant de $Sims(s, d)$ (voir Gelly & Silver (2007)).

  end switch
end for
Retourner $\arg\max_d\ score(d)$.

L'algorithme AMAF a été dérivé de Gelly & Silver (2007). Les simulations d'AMAF sont créées par permutation de coup dans la simulation réelle. Un progrès important, le "Progressive Widening", de l'algorithme 3 consiste à considérer uniquement les $K(n)$ meilleurs coups (selon une heuristique choisie) à la n-ième simulation d'un noeud donné. Avec $K(n)$ une application non-décroissante de $\mathbb{N}$ dans $\mathbb{N}$.

Algorithm 4 Algorithme de planification Monte-Carlo plus détaillé.

Initialiser T à un noeud unique, représentant l'état courant. T est un arbre de positions.
while Temps restant > 0 **do**
  Simuler une partie jusqu'à une feuille (une position) L de T (à l'aide de l'algorithme de bandit jusqu'à ce qu'on arrive à une feuille, voir algorithme 3).
  Choisir un fils (un successeur) L' de L, éventuellement à l'aide de connaissances a priori.
  Simuler une partie (voir algorithme 2) depuis la position L' jusqu'à la fin de la partie.
  Développer l'arbre : ajouter L' en tant que fils de L dans T.
  Mettre à jour les statistiques dans tout l'arbre.
end while
Retourner le coup qui a été simulé le plus souvent à partir de la racine.

Malheureusement, plusieurs problèmes surviennent : premièrement, la partie Monte Carlo n'est pas encore complètement comprise. En effet, de nombreuses discussions autour de ce sujet ont été publié sur la mailing list computer-go, mais personne n'est encore parvenu à proposer un critère pour déterminer sans expérimentations coûteuses de l'algorithme BBCMP, ce qu'est un bon simulateur de Monte Carlo. Ainsi, utiliser un bon joueur de go, tel qu'un programme comme GnuGo, dans ce but, réduit fortement les performances du programme, même à nombre de simulations fixées. Une explication intuitive de ce phénomène pourrait être qu'un simulateur de Monte Carlo fortement biaisé mène à une estimation incorrecte de l'espérance de victoire pour les noeuds. Deuxièmement, la partie Bandit est encore trouble. Des implémentations utilisent encore la formule UCB Auer *et al.* (2002) mais avec une petite constante d'exploration, ainsi les coups avec le meilleur résultats empiriques sont utilisés, indépendamment du nombre de simulations, jusqu'à ce que la probabilité de victoire deviennent plus petite que la probabilité de victoire des noeuds

Sans livre d'ouverture	livre standard	livre cosmique
55.5 % ± 1%	53.5 % ± 1%	56.6% ± 1.3%

TAB. 1 – Résultats expérimentaux pour les livres d'ouverture. L'adversaire est gnugo3.6, niveau par défaut. MoGo joue en tant que noir.

non parcourus. Nous allons présenter à présent (i) les améliorations pour la partie arbre de BBMCP, voir l'algorithme 3 et (ii) les améliorations de la partie Monte-Carlo, voir l'algorithme 2.

2 Livres d'ouvertures

Les livres d'ouvertures constituent la méthode la plus simple d'introduire de la connaissance experte au sein de l'algorithme BBMCP appliqué au jeu de go. En particulier, il est connu que BBMCP fait de mauvais choix en début de partie. Ceci semble simple à améliorer a priori. Nous avons donc développé (i) un livre d'ouvertures standard basé sur des parties de professionnels (ii) un ensemble de parties spécifiques proches du style de jeu de MoGo, privilégiant un contrôle rapide du centre du plateau, appelé dans la littérature "style cosmique". Les résultats expérimentaux sont présentés dans le Tableau 1, avec 5000 simulations par coup.

L'amélioration est faible, voire négative dans le cas des livres d'ouvertures classiques.

Nous avons également généré des livres d'ouvertures automatiquement en faisant jouer MoGo contre lui-même. Ceux-ci se sont révélés efficaces (approximativement 60% contre la version initiale) mais uniquement contre une version de MoGo avec un niveau modéré. En effet, ces mouvements obtenus par des parties de MoGo avec des longs temps de simulation contre un MoGo aux décisions bruitées afin de diversifier les situations d'ouvertures, ne sont pas adaptés à la version parallèle de MoGo. En effet, nous aurions besoin de milliers d'heures de calcul pour générer, par ce type de parties, un livre d'ouverture suffisamment performant pour la version parallèle de MoGo, sans quoi la taille du livre d'ouverture resterait trop restreinte.

3 Amélioration des simulations dans l'arbre

Dans la partie arbre, le but n'est pas de trouver de bons coups mais de trouver les coups qui valent la peine d'être explorés. Dans beaucoup de programmes, cela est fait par des algorithmes de type UCT, i.e. en choisissant le coup tel que la quantité suivante soit maximale :

$$\underbrace{w(move)/n(move)}_{Exploitation} + C \underbrace{\sqrt{\log(\sum_{m \in M} n(m))/n(move)}}_{Exploration} \quad (1)$$

où :
- $n(move)$ est le nombre de fois où le coup a été exploré dans cette situation ;
- $w(move)$ est le nombre de fois où le coup a été essayé et a conduit à une victoire ;
- M est l'ensemble des coups possibles dans cette situation.

Une amélioration classique consiste à remplacer Eq. 1 par certaines versions améliorées, typiquement UCB-Tuned Auer *et al.* (2002), c'est une solution générale qui n'est pas spécifique aux jeux et qui peut être appliquée à la planification. Dans le cas des jeux, des formules comme les valeurs AMAF dans l'algorithme 3 peuvent être utilisées pour biaiser le terme d'exploration vers les coups qui paraissent intéressants selon les statistiques "AMAF" Bruegmann (1993). Nous voulons introduire ici un biais par dessus ce biais en ajoutant des simulations AMAF virtuelles.

La suppression de l'arbre des coups vraissemblablement mauvais a été expérimenté ; cependant, de tels suppressions sont dangereuses au Go, car les exceptions aux règles sont fréquentes. Nous nous contenterons donc d'introduire un biais en faveur des coups "intéressants" en ajoutant des simulations virtuelles : au lieu de créer de nouveaux noeuds dans l'arbre sans aucune simulation, nous introduisons artificiellement des gains pour des coups qui sont supposés être bons ou, plus précisément, pour des coups qui sont supposés valoir la peine d'être explorés. Inversement, nous introduirons des pertes pour les coups supposés faibles. D'autres solutions pour inclure des connaissances expertes existent :

– Progressive widening Coulom (2007) ou progressive unpruning Chaslot *et al.* (2007) : le coup joué est choisi *parmi les n coups* ayant le score le plus élevé (eq. 1), n dépendant de connaissances expertes ;
– Progressive bias Chaslot *et al.* (2007) : la formule 1 est modifiée par un terme $+heuristic\,value(move)/n(move)$;
– First play urgency Wang & Gelly (2007) : le score représenté par la formule 1 est utilisé pour les coups m tels que $n(m) > 0$, et une valeur à priori dépendant des connaissances expertes est utilisée pour les autres coups ;
– MoGo incorpore un mélange de "Progressive bias" et de "Progressive widening" par l'ajout d'un bonus diminuant avec le nombre de simulations pour des coups proches au dernier coup joué. La distance est conçue à travers l'expertise Go : c'est une distance topologique qui suppose la distance 1 entre toutes les pierres d'une chaîne, quoiqu'elle peut être la taille de cette chaîne.

Divers éléments sur des connaissances expertes déjà publiés (e.g. Coulom (2007); Bouzy & Chaslot (2005)) incluent :
– les formes : dans Coulom (2007); Bouzy & Chaslot (2005), les formes sont générées automatiquement à partir d'un ensemble de données. Nous présentons ci-dessous les connaissances expertes introduites sur les formes ;
– les coups de capture (en particulier, le coup voisin à une nouvelle chaîne en atari), chaîneion (notamment le shisho), l'évitement de se mettre en atari, l'atari (en particulier quand il y a un ko), la distance au bord (distance optimale $= 3$ en 19x19 Go), une petite distance au coup précédemment joué, une petite distance à l'avant-dernier coup, et aussi la caractéristique reflétant le fait que la probabilité (selon les simulations de Monte-Carlo) d'un emplacement à être de sa propre couleur à la fin de la partie est $\simeq 1/3$.

Les outils suivants sont utilisés dans nos implémentations en 19x19, et améliorent les résultats de l'heuristique AMAF :
– La ligne de territoire (i.e. ligne numéro 3) : 1.333 gains dans les simulations AMAF sont ajoutés ;
– La ligne de la mort (i.e. première ligne) : 1.333 pertes dans les simulations AMAF sont ajoutés ;
– Peep-connect (ie. connecter deux chaînes quand l'adversaire menace de couper) : 1 gain dans les simulations AMAF est ajouté ;
– Hane (un coup qui "contourne" une ou plusieurs pierres de l'adversaire) : 1 gain dans les simulations AMAF est ajouté ;
– Connect : 1 gain dans les simulations AMAF est ajouté ;
– Wall : 0.666 gains dans les simulations AMAF sont ajoutés ;
– Bad Kogeima(même motif que le mouvement du cavalier aux échecs) : 0.5 pertes dans les simulations AMAF sont ajoutés ;
– Empty triangle (trois pierres de même couleur formant un triangle sans pierre adverse dans l'angle formé) : 1 perte dans les simulations AMAF est ajoutée.

Ces formes sont illustrées sur la Figure 1. Avec un réglage naïf des paramètres à la main, elles fournissent 63.9±0.5 % de victoires contre la version sans ces améliorations. Nous sommes optimistes sur le fait que le réglage des paramètres améliorera considérablement les résultats. D'ailleurs, dans les premiers développements de MoGo, certains bonus de "coupe" sont déjà inclus (i.e., les avantages de jouer à des emplacements qui vérifie des motifs de "coupe", i.e. des motifs pour lesquels un emplacement empêche l'adversaire de connecter deux groupes).

4 Amélioration des simulations Monte-Carlo (MC)

On ne sait pas encore comment créer un bon simulateur Monte-Carlo pour BBMCP dans le cas du Go. De nombreuses personnes ont essayé d'améliorer le simulateur en augmentant sa force en tant que joueur indépendant, mais il a clairement été prouvé dans Gelly & Silver (2007) que ce n'est pas un bon critère : un simulateur MC_1 qui joue beaucoup mieux qu'un simulateur MC_2 peux conduire à de très mauvais résultats une fois utilisé avec BBMCP et ceci même en comparant à nombre de simulations identique. Certains simulateurs ont été appris à partir de base de données Coulom (2007), mais les résultats sont fortement améliorés par un réglage manuel des paramètres. Les raisons expliquant qu'un simulateur MC va être efficace étant encore inconnues, il est nécessaire d'expérimenter toute modification de manière intensive afin de la valider.

De nombreuses formes sont définies dans Bouzy (2005); Wang & Gelly (2007); Ralaivola *et al.* (2005).

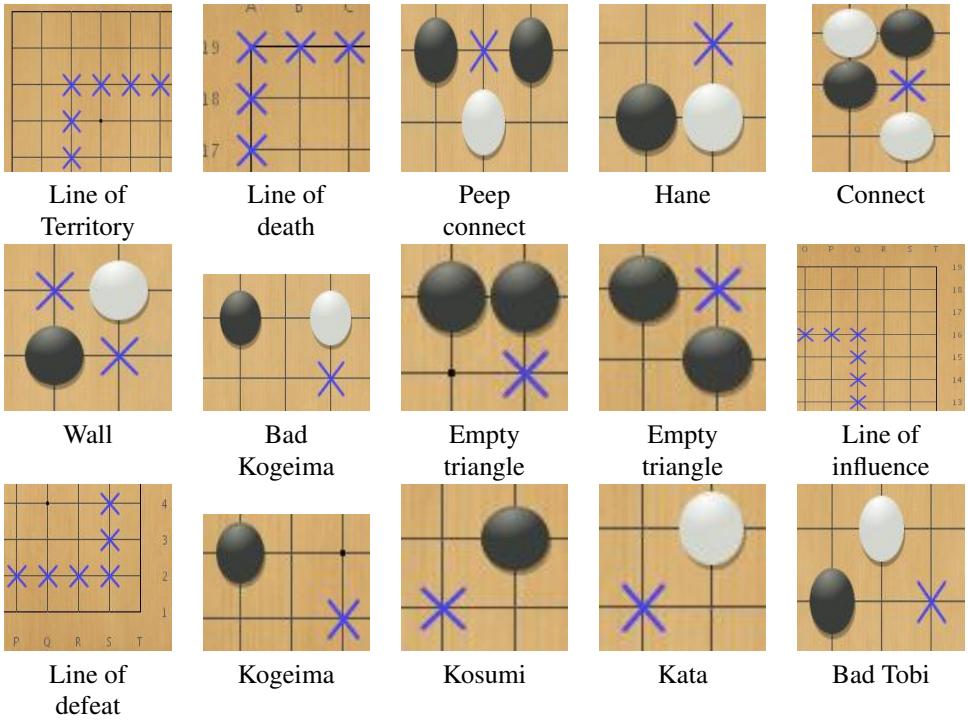


FIG. 1 – Nous présentons ici les motifs exacts pour lesquels s'appliquent les bonus /malus. Dans tous les cas, les motifs sont présentés avec trait aux Noirs : les formes s'appliquent pour un coup Noir à une des croix. Les motifs avec couleurs inversées s'appliquent bien sûr aux Blancs. Dans le cas du Bad Kogeima, il ne doit pas y avoir de pierres noires autour de la croix pour que le bonus s'applique. Dans le cas du Kosumi et du Kogeima, il ne doit pas y avoir de pierres blanches entre la pierre noire et la croix. Le motif "seul sur la première ligne" s'applique seulement si les 5 emplacements autour du coup considéré sont vides. Dans le cas du bad tobi, il ne doit pas y avoir de pierre noire parmi les 8 emplacements autour de la croix. Dans le cas du Hane, la pierre blanche ne doit avoir aucune pierre amie parmi les 4 emplacements voisins.

Wang & Gelly (2007) utilise des patterns et des connaissances expertes comme expliqué dans l'algorithme 2. Nous présentons ci-dessous nouvelles améliorations.

4.1 fillboard : on remplit les espaces vides du plateau en premier

Le principe de cette modification est de jouer en priorité aux emplacements du plateau où il n'y a pas encore de pierre autour. L'idée est de jouer de manière plus uniforme dans ces zones. Essayer chaque emplacement du plateau serait trop coûteux en temps, c'est pourquoi quoi nous utilisons la procédure suivante. Une position du plateau est choisie au hasard ; si les 8 emplacements contiguës sont vides alors le coup est joué, sinon on essaye les $N-1$ positions suivantes du plateau. Cette modification introduit plus de diversités dans les simulations. Ceci est dû au fait que le simulateur MC de MoGo utilise beaucoup de patterns. Or quand un pattern correspond, le coup est joué. Les simulations n'ont donc qu'un petit nombre de manières de commencer, en particulier en début de partie où il n'y a que peu de pierres sur le plateau. Cette modification intervenant avant la détection de patterns, elle permet d'obtenir des simulations plus diversifiées (voir figure 2 (gauche)). L'algorithme détaillé est présenté dans l'algorithme 5.

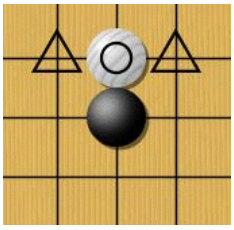
Les expériences sont présentées dans la figure 2 (droite).

4.2 Le problème du "Nakade"

. Une faiblesse connue de MoGo, ainsi que de beaucoup de programme BBMCP, est que les "nakade" ne sont pas gérés correctement. Nous utiliserons le terme *nakade* pour désigner une situation dans laquelle un groupe encerclé possède un unique espace interne enfermé à partir duquel le joueur ne pourra pas créer deux yeux si l'adversaire répond correctement. Le groupe est donc mort mais le simulateur MC utilisé dans

Algorithm 5 Algorithme permettant de choisir un coup dans les simulations MC, comprenant l'amélioration "fillboard". Nous avons également expérimenté avec une contrainte de 4 emplacements libres au lieu de 8 mais les résultats furent décevants.

if le dernier coup est un atari **then**
  Sauver les pierres en atari.
else
  partie "Fillboard".
  for $i \in \{1, 2, 3, 4, \ldots, N\}$ **do**
    Tirer une position x au hasard sur le plateau.
    Si x est un emplacement vide et que les 8 emplacements autour sont également vide, jouer x (sortir).
  end for
  Fin de la partie "fillboard".
  if il y a un emplacement vide parmi les 8 emplacements autour du dernier coup qui correspond à un pattern **then**
    Jouer au hasard sur l'un de ces emplacements (sortir).
  else
    if il y a un coup qui capture des pierres **then**
      Capturer les pierres (sortir).
    else
      if il reste des coups légaux **then**
        Jouer au hasard un coup légal (sortir).
      else
        Renvoyer "passe".
      end if
    end if
  end if
end if



9x9 board		19x19 board	
Nb de simulations par coup ou temps/coup	Taux de succès	Nb de simulations par coup ou temps/coup	Taux de succès
10 000	52.9 % ± 0.5%	10000	49.3 ± 1.2 %
5s/coup, 8 coeurs	54.3 % ± 1.2 %	5s/coup, 8 coeurs	77.0 % ± 3.3 %
100 000	55.2 % ± 1.4 %	100 000	73.7 % ± 2.9%
200 000	55.0 % ± 1.1 %	200 000	78.4 % ± 2.9 %

FIG. 2 – Gauche : perte de diversité quand on n'utilise pas l'option "fillboard" : la pierre blanche est la dernière pierre jouée, une simulation MC pour le joueur noir commencera obligatoirement par l'une des cases marquée d'un triangle. Droite : Résultats associés à la modification "fillboard". Comme la modification induit un surcoût computationnel, les résultats sont meilleurs pour un nombre de simulations par coup fixé ; cependant, l'amélioration est clairement significative. Le surcoût computationnel est réduit lorsque l'on utilise plusieurs coeurs : en effet, il y a moins de concurrence pour l'accès mémoire lorsque les simulations sont plus longues. C'est pourquoi la différence en temps entre des simulations simples et complexes se réduit avec le nombre de coeurs. Cet élément montre l'intérêt accru des simulations complexes dans le cadre de la parallélisation.

la version de MoGo de référence (Algorithme 2) va estimer que le groupe vit avec une forte probabilité. En effet, le simulateur ne répondra pas les coups considérés comme évident pour un joueur et peut donc conduire à la survie du groupe. L'arbre va donc sous estimer le danger correspondant à cette situation et éventuellement se développer dans cette direction. La partie MC doit donc être modifiée pour que la probabilité de gagner prenne en compte ces situations de *nakade* .

Il est intéressant de noter que MoGo étant principalement développé en jouant contre lui-même, cette faiblesse n'est apparue qu'une fois trouvée par un humain (voir le mail de D.Fotland "UCT and solving life

and death" sur la mailing list computer-go).

Il serait théoriquement possible d'encoder dans les simulations MC un vaste ensemble de comportements relatifs au *nakade* , mais cette approche a deux faiblesses : (i) cela serait coûteux en temps et les simulations doivent être rapides (ii) changer le comportement du simulateur MC de manière trop abrupte conduit généralement à des résultats décevants. C'est pourquoi la modification a été créée : si on trouve un ensemble d'exactement 3 intersections libres entourées de pierres adverses, on joue au milieu (le point vital). Cela permet de fortement diminuer la probabilité de survie de la plupart des situations de *nakade* . Le nouvel algorithme est présenté dans l'algorithme 6.

Algorithm 6 Nouveau simulateur MC, permettant d'éviter le problème de *nakade* .

if le dernier coup est un atari **then**
  Sauver les pierres en atari.
else
  Début de la modification *nakade*
  for x parmis l'un des 4 emplacements libres autour du dernier coup joué **do**
    if x fait parti d'un trou de 3 emplacements contigus entourés de pierres adverses **then**
      Jouer au centre du trou (sortir).
    end if
  end for
  Fin de la modification *nakade*
  partie "Fillboard".
  for $i \in \{1, 2, 3, 4, \ldots, N\}$ **do**
    Tirer une position x au hasard sur le plateau.
    Si x est un emplacement vide et que les 8 emplacements autour sont également vide, jouer x (sortir).
  end for
  Fin de la partie "fillboard".
  if il y a un emplacement vide parmi les 8 emplacements autour du dernier coup qui correspond à un pattern **then**
    Jouer au hasard sur l'un de ces emplacements (sortir).
  else
    if il y a un coup qui capture des pierres **then**
      Capturer les pierres (sortir).
    else
      if il reste des coups légaux **then**
        Jouer au hasard un coup légal (sortir).
      else
        Renvoyer "passe".
      end if
    end if
  end if
end if

Cette approche est validée par deux expériences différentes : (i) des positions connues où la version de MoGo de référence ne choisit pas le bon coup (figure 3) (ii) des parties nouveau MoGo contre MoGo de référence (table 2).

4.3 Améliorations cumulées

Les deux améliorations de la partie Monte-Carlo sont indépendantes. Le tableau suivant montre le pourcentage de victoire de chaque amélioration et des deux cumulées contre la version de référence avec 10 000 simulations par coup.

Amélioration	Taux de succès
fillboard	52.9 % $\pm$ 0.5%
nakade	52.8 % $\pm$ 0.5 %
les deux	57.2 % $\pm$ 1.0 %

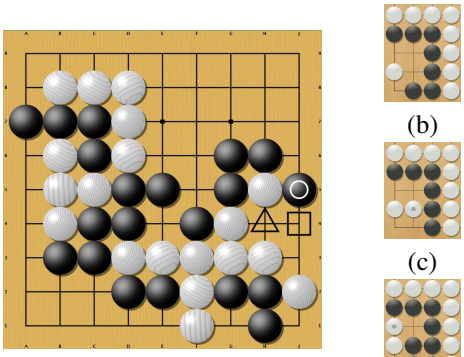


FIG. 3 – figure (a) (une partie réelle jouée et perdue par MoGo), MoGo (blanc) sans la modification pour les *nakade* joue H4 ; noir répond J4 et le groupe F1 est mort (MoGo perd). Après la modification, MoGo joue J4 qui est le bon coup car il permet de vivre après un ko. Les exemples (b), (c) et (d) sont des situations similaires dans lesquelles MoGo ne réalise pas que son groupe est mort. Dans chacun des cas, la modification permet de résoudre le problème.

Nb de simulations par coup	Taux de succès	Nb de simulations par coup	Taux de succès
plateau 9x9		plateau 19x19	
10000	52.8 % ± 0.5%		
100000	55.6 % ± 0.6 %	100 000	53.2 % ± 1.1%
300000	56.2 % ± 0.9 %		
5s/coup, 8 coeurs	55.8 % ± 1.4 %		
15s/coup, 8 coeurs	60.5 % ± 1.9 %		
45s/coup, 8 coeurs	66.2 % ± 1.4 %		

TAB. 2 – Validation expérimentale de la modification *nakade* : MoGo modifié contre MoGo de référence. Il apparaît que plus le nombre de simulations est important (ce qui est directement lié au niveau), plus l'impact de la modification est important.

5 Conclusion

Nos conclusions sont les suivantes :

- Les ouvertures sont un problème difficile : utiliser des ouvertures de joueurs professionnels est une mauvaise idée. Le niveau et le style des premiers coups doivent être consistants avec ceux des coups suivants. Générer ces ouvertures en faisant jouer MoGo contre lui-même est une approche stable et efficiente. Cependant, pour la version parallèle de MoGo, nous aurions besoin d'un cluster aussi gros que celui que nous utilisons dans les parties réelles, et cela pour un grand nombre de positions. Malheureusement, nous n'avons pas accès à un si grand nombre d'heures-CPU.
- Introduire de l'expertise Go en réduisant la largeur de l'arbre est dangereux, et donne des résultats négatifs, notamment lorsque le nombre de simulations est élevé. Une autre approche qui consiste à augmenter le nombre initial de simulations gagnées ou perdues est fiable et améliore grandement le programme.
- Nous n'avons pas d'idée de critères formels permettant de juger la partie Monte-Carlo de l'algorithme. Nous utilisons des modifications qui améliorent sa performance, mais un il est intéressant de noter que la variance ne devrait pas être trop réduite. En augmentant le nombre possible de coups, la modification "fillboard" permet un nombre de futurs possibles plus large. Les patterns permettent d'éviter les simulations absurdes, mais cela peut aussi réduire fortement la variance du résultat.
- Se concentrer sur des contre-exemples est efficace. La solution au problème du nakade a été trouvée en observant le comportement de la partie Monte-Carlo dans des situations de nakade. De ce point de vue, des ensembles de tests tels ceux proposés par M. Yamato (`http://computer-go.org/pipermail/computer-go/2008-April/014773.html`) sont un moyen pratique d'amélio-

rer le programme.

– Dans le futur, il sera important de (i) réduire l'agressivité de BBMCP en changeant la partie Monte-Carlo, de la même manière que le "style cosmic" avait été réduit par la modification "fillboard", (ii) améliorer MoGo dans les situations de ko, probablement en rajoutant un biais dans l'arbre, (iii) introduire plus de diversité dans les simulations, par exemple en introduisant différentes formes de Monte Carlo, ou en faisant dépendre le coup joué dans la simulation de coups antérieurs au dernier coup joué et (iv) développer des bases de données d'ouvertures meilleures et plus exhaustives.

Remerciements

Nous remercions Rémi Munos, Yizao Wang, Rémi Coulom, Guillaume Chaslot, Tristan Cazenave, Jean-Yves Audibert, David Silver, Martin Mueller,KGS, Cgos, ainsi que toute la mailing list computer-go pour toutes les discussions intéressantes que nous avons pu avoir. Un grand merci à la fédération française de Go et à Recitsproque pour avoir organisé un match officiel contre un humain de haut niveau ; un grand merci également à tous les joueurs de la fédération française de Go qui ont accepté de jouer et de commenter des parties test contre MoGo, et à Catalin Taranu, joueur professionnel 5e dan, pour ses commentaires après le challenge IAGO. Nous remercions Antoine Cornuejols pour ses commentaires très utiles sur la première version de cet article.

Références

AGRAWAL R. (1995). The continuum-armed bandit problem. *SIAM J. Control Optim.*, **33**(6), 1926–1951.

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

BANKS J. S. & SUNDARAM R. K. (1992). Denumerable-armed bandits. *Econometrica*, **60**(5), 1071–96. available at http ://ideas.repec.org/a/ecm/emetrp/v60y1992i5p1071-96.html.

BOUZY B. (2005). Associating domain-dependent knowledge and monte carlo approaches within a go program. In K. CHEN, Ed., *Information Sciences, Heuristic Search and Computer Game Playing IV*, volume 175, p. 247–257.

BOUZY B. & CHASLOT G. (2005). Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 go. *In G. Kendall and Simon Lucas, editors, IEEE 2005 Symposium on Computational Intelligence in Games, Colchester, UK*, p. 176–181.

BRUEGMANN B. (1993). Monte carlo go. *Unpublished*.

CHASLOT G., WINANDS M., UITERWIJK J., VAN DEN HERIK H. & BOUZY B. (2007). Progressive strategies for monte-carlo tree search. In P. WANG & OTHERS, Eds., *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, p. 655–661 : World Scientific Publishing Co. Pte. Ltd.

COQUELIN P.-A. & MUNOS R. (2007). Bandit algorithms for tree search. In *Proceedings of UAI'07*.

COULOM R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*.

COULOM R. (2007). Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*.

DANI V. & HAYES T. P. (2006). Robbing the bandit : less regret in online geometric optimization against an adaptive adversary. In *SODA '06 : Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, p. 937–943, New York, NY, USA : ACM Press.

GELLY S. & SILVER D. (2007). Combining online and offline knowledge in uct. In *ICML '07 : Proceedings of the 24th international conference on Machine learning*, p. 273–280, New York, NY, USA : ACM Press.

KOCSIS L. & SZEPESVARI C. (2006). Bandit-based monte-carlo planning. In *ECML'06*, p. 282–293.

LAI T. & ROBBINS H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, **6**, 4–22.

RALAIVOLA L., WU L. & BALDI P. (2005). Svm and pattern-enriched common fate graphs for the game of go. In *Proceedings of ESANN 2005*, p. 485–490.

WANG Y. & GELLY S. (2007). Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*, p. 175–182.

Méthode pour la Planification de Trajectoires Garanties

Sébastien LENGAGNE[1,2,3], Nacim RAMDANI[2,3], Philippe FRAISSE[1,3,4]

[1] Équipe-projet DEMAR
http://www-sop.inria.fr/demar/

[2] Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier : LIRMM-UMR 5506
161 rue Ada 34392 Montpellier Cedex 5 - France
http://www.lirmm.fr

[3] Institut National de Recherche en Informatique et en Automatique INRIA
http://www.inria.fr

[4] Université Montpellier 2 : UM-2
http://www.univ-montp2.fr

Résumé :
Dans cet article, nous présentons une méthode générale de planification de trajectoires sécurisées, appliquée à l'optimisation d'un pas dans le plan sagittal pour le robot humanoïde HOAP-3. La planification de trajectoires sécurisées utilise les mêmes algorithmes d'optimisation que les méthodes de planifications classiques, mais remplace la discrétisation par grille par une discrétisation garantie basée sur l'analyse par intervalles. Les résultats expérimentaux obtenus avec un modèle 2-D à six degrés de liberté du robot humanoïde HOAP-3 prouve que la méthode classique peut engendrer la violation de contraintes alors que la méthode proposée garantit le respect des contraintes sur tout le mouvement assurant ainsi l'équilibre et l'intégrité du robot.

Mots-clés : planification de trajectoires, contraintes garanties, optimisation, discrétisation, analyse par intervalle, robot humanoïde,sécurité, boucle ouverte.

1 Introduction

Les travaux, présentés dans cette communication, s'inscrivent dans le cadre des activités du projet INRIA-DEMAR (DÉambulation et Marche ARtificielle) (4) qui a pour but la restauration du mouvement, et notamment la déambulation, de patients paraplégiques. Dans le contexte de notre étude, contrairement à la réalisation de mouvements pour des robots humanoïdes, nous ne disposons pas des capteurs nécessaires au développement d'une boucle de commande. Par conséquent, l'exécution de mouvements sera réalisée en boucle ouverte. Il faut également souligner que nous devons absolument garantir l'intégrité des patients paraplégiques. Il est donc nécessaire de développer une méthode générale de planification de trajectoires sécurisées garantissant le respect des contraintes pour toute la durée du mouvement. Afin de valider notre méthode, nous l'expérimenterons sur le robot humanoïde HOAP-3.

La planification de trajectoires a été le but de nombreuses recherches de la communauté robotique humanoïde. Pour des systèmes complexes tels que les robot humanoïdes, la planification de trajectoires doit prendre en compte un nombre important de contraintes, ce qui augmente les temps de calcul. C'est pour cette raison que la planification de trajectoires se fait, dans le domaine de la robotique, généralement hors ligne dans le but de créer des bases de données de mouvements (1) qui seront exécutés en ligne afin de palier au problème du temps de calcul. La planification de trajectoires est un domaine vaste qui rassemble le problème de la locomotion d'acteurs virtuels dans des environnements encombrés (18), la génération de mouvement de coup de pied (10) ou le calcul de trajectoires pour des robots manipulateurs (13). Tous ces problèmes peuvent se ramener à un problème d'optimisation sous contraintes.

Dans la plupart des cas, les contraintes inhérentes au système doivent être satisfaites tout au long de la trajectoire générée. En pratique, cependant, ces contraintes sont discrétisées en ne considérant leur valeur que pour des instants appartenant à une grille temporelle. Par conséquent l'algorithme d'optimisation produira un mouvement qui satisfera les contraintes aux instants consiédérés, sans garantir la non-violation entre

deux instants. Cette absence de garantie peut provoquer des violations de contraintes qui peuvent avoir des conséquences désastreuses pour l'intégrité du système.

Pour apporter une solution à ce problème, nous développons, dans cette communication, une méthode qui se base sur la discrétisation garantie des contraintes, déjà testé pour le calcul d'un pas pour un système à deux degrés de liberté (8). Nous étendons ces travaux à la planification de trajectoires sécurisées pour des systèmes plus complexes. Nous montrons comment générer un pas optimal dans le plan sagittal pour le robot humanoïde HOAP-3. Nous nous intéressons à la discrétisation temporelle, mais notre méthode peut être appliquée à une discrétisation à N-dimensions. Le principe de base est de considérer les contraintes sur un ensemble d'intervalles de temps au lieu de ne prendre en compte qu'un ensemble d'instants discrets. Pour cela, nous utilisons l'analyse par intervalle (11) qui a déjà été utilisée pour résoudre des problèmes de chemin garanti évitant les auto-collisions (2). Nous illustrons notre méthode de planification de trajectoires sécurisées avec un modèle 2-D de robot humanoïde, mais elle peut être généralisée à des modèles 3-D.

Ce document est structuré de la manière suivante : la section 2 introduit le modèle 2-D du robot humanoïde utilisé pour la génération de pas optimaux dans le plan sagittal. La section 3 présente le problème d'optimisation ainsi que la discrétisation par grille couramment utilisée. La section 4 introduit l'analyse par intervalle utilisée dans le cadre de la planification de trajectoires sécurisées présentées en section 5. La comparaison des résultats dans la section 6 précède la section 7 où les travaux futurs sont exposés.

2 Modélisation

Pour planifier un mouvement, il est nécessaire de connaître le système grâce à un modèle mathématique La planification de trajectoires optimales doit tenir compte de plusieurs contraintes inhérentes au système qui sont obtenues grâce à un modèle. Dans notre cas, nous nous intéressons à la planification de mouvements pour le robot humanoïde HOAP-3 dans le sagittal et considérons que l'équilibre dans le plan frontal peut être assuré en utilisant, par exemple, la méthode présentée dans (3). Nous simplifions le robot humanoïde HOAP-3 (système 3D) en un système plan avec 6 degrés de liberté, en supposant que les membres supérieurs sont équivalents à une masse située au niveau du bassin (cf. Fig 1).

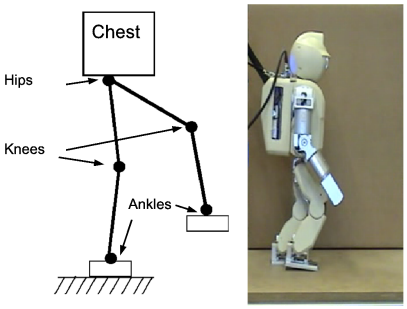


FIG. 1 – modèle plan du robot HOAP-3 pour la planification d'un pas dans le plan sagittal

Nous définissons le mouvement à travers un vecteur de paramètres **X** qui contient des coefficients pondérateurs $P_{j,k}$ ainsi que la durée du mouvement T_f.

$$\mathbf{X} = [P_{0,0}; P_{0,1}; ...; T_f]^T \tag{1}$$

À Partir de ce vecteur de paramètres nous allons calcules les positions, vitesses et accélérations articulaires. Puis, partant de ces valeurs, nous déterminerons les couples articulaires grâce au modèle dynamique inverse qui nous fournira également des informations pour calculer une grandeur représentative de l'équilibre du robot.

2.1 Variables articulaires

Chaque variable articulaire est calculée à partir de N_s coefficients pondérateurs appartenant au vecteur **X**.

$$q_j(t) = \sum_{k=1}^{N_s} P_{j,k} \times b_k(t) \tag{2}$$

Les fonctions $b_k(t)$ forment une base de fonctions représentée sur la Figure 2 Les vitesses et accélérations

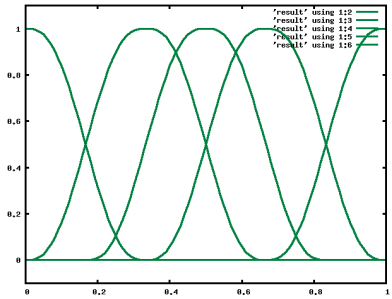


FIG. 2 – Représentation des fonctions de base $b_k(t)$ pour $N_s = 5$

articulaires sont obtenues par dérivation :

$$\dot{q}_j(t) = \sum_{k=1}^{N_s} P_{j,k} \times \dot{b}_k(t) \qquad \ddot{q}_j(t) = \sum_{k=1}^{N_s} P_{j,k} \times \ddot{b}_k(t) \tag{3}$$

2.2 Modèle dynamique

Les équations du modèle dynamique permettent de calculer les couples articulaires $\Gamma_{(t)}$ à partir des positions, vitesses et accélérations articulaires et des efforts extérieurs $f_{e(t)}$ (considérés nuls dans notre cas). Nous utilisons la méthode Newton-Euler (7) qui permet un calcul rapide des couples articulaires ainsi que des efforts au niveau du pied de support.

$$\left\{ \begin{array}{c} \Gamma_{(t)} \\ F_{pied} \end{array} \right\} = NE(q_{(t)}, \dot{q}_{(t)}, \ddot{q}_{(t)}, f_{e(t)}) \tag{4}$$

2.3 Équilibre

Le calcul du Zero Moment Point (ZMP) nous informe sur l'équilibre des robots humanoïdes pendant le mouvement. (17) définit le ZMP comme le point sur la surface de contact ou le moment est nul (cf. Fig 3). Si ce point reste dans la surface de support le robot conserve son équilibre, sinon il est dans un état de déséquilibre qui peut entraîner la chute.

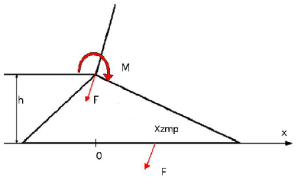


FIG. 3 – Représentation du *ZMP*

La valeur du ZMP dépend des positions, vitesses et accélérations articulaires, ainsi que des efforts extérieurs :

$$ZMP(t) = f(q(t), \dot{q}(t), \ddot{q}(t), f_{e(t)}). \tag{5}$$

2.4 Contraintes du mouvement

Dans les sections suivantes, nous considérerons qu'un mouvement optimal doit obligatoirement respecter les contraintes de butées articulaires ainsi que de vitesses articulaires limites.

$$\forall i, \forall t \qquad q_{i,min} \leq q_i(t) \leq q_{i,max} \tag{6}$$

$$\forall i, \forall t \qquad \dot{q}_{i,min} \leq \dot{q}_i(t) \leq \dot{q}_{i,max} \tag{7}$$

Afin d'obtenir un mouvement qui préserve l'équilibre, nous prendrons également en compte les limites du ZMP.

$$\forall t \qquad ZMP_{min} \leq ZMP(t) \leq ZMP_{max} \tag{8}$$

3 Planification de trajectoires sous contraintes

Nous avons donc définis le modèle mathématique de notre système. Nous pouvons donc procéder à la planification des trajectoires articulaires qui définissent le mouvement (coup de pied, accomplissement d'un pas).

3.1 Problème d'optimisation

Le problème de planification de trajectoires se résume souvent à un problème d'optimisation qui consiste à trouver le vecteur de paramètres optimal $\mathbf{X}$ qui satisfait :

$$\min \int_0^T F(\mathbf{X},t)dt \tag{9}$$

$$\text{subject to :} \quad \begin{array}{rr} \forall i, \forall t \in [0,T] & g_i(\mathbf{X},t) \leq 0 \\ \forall j & h_j(\mathbf{X}) = 0 \end{array} \tag{10}$$

où :
- F est la fonction objectif à minimiser,
- g_i est un ensemble de contraintes inégalités,
- h_j est un ensemble de contraintes égalités.

Le problème posé (Eq. 9-10) est un problème de programmation semi-infinie (6) car il dispose d'un nombre fini de paramètres alors que le domaine des contraintes est infini. (14) décrit trois méthodes différentes de discrétisation des problèmes de programmation semi-infinie qui permettent d'approximer le problème semi-infini en un problème fini. L'idée principale de ces méthodes est de déterminer une grille temporelle afin de résoudre le problème d'optimisation aux intersections de la grille. Une fois le problème résolu, la grille est modifiée pour obtenir un résultat plus précis.

Cependant une grille de discrétisation n'assure pas le respect des contraintes pour des points non contenus dans cette grille. Le paragraphe suivant montre une discrétisation par grille et met en lumière la violation possible de contraintes.

3.2 Discrétisation par grille

Dans la plupart des cas, discrétiser consiste à calculer les contraintes pour un ensemble discret de valeur du temps (16). Par conséquent, l'équation (10) devient :

$$\begin{array}{rr} \forall i, \forall t_k \in \mathbf{T} & g_i(\mathbf{X},t_k) \leq 0 \\ \forall j & h_j(\mathbf{X}) = 0 \end{array} \tag{11}$$

avec $\mathbf{T} = \{t_0 = 0, t_1, ..., t_{N-1}, t_N = T\}$.

Le problème semi-infini (9-10), défini comme $\forall t \in [0,T]$, est considéré comme un problème fini : $\forall t_k \in \{0, t_1, ..., t_{N-1}, T\}$ pour lequel les contraintes ne seront satisfaites uniquement aux instants pris en compte.

3.3 Violation des contraintes

En utilisant une discrétisation par grille l'algorithme d'optimisation produira une solution qui satisfera les contraintes seulement pour les instants discrets $\{t_0, t_1 ..., t_{N-1}, t_N\}$, mais aucune garantie n'est donnée concernant la validité des contraintes pour des instants non considérés. Afin de mettre en évidence ce phénomène nous avons utilisé cette méthode de discrétisation pour planifier le mouvement d'un pas pour le robot humanoïde HOAP-3. Nous considérons les contraintes des équations (6,7,8) et voulons minimiser la durée du mouvement.

La Figure 4 montre l'évolution du ZMP pour un pas optimisé en considérant 10 instants de discrétisation. Dans ce cas pour que le robot conserve son équilibre il faut que la valeur du ZMP reste dans l'intervalle

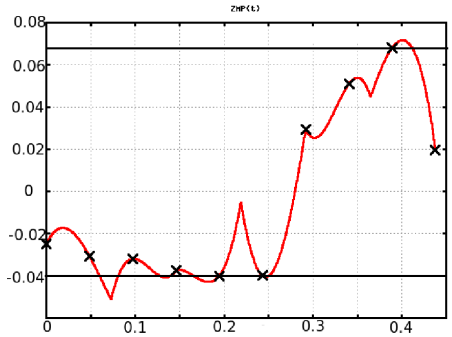
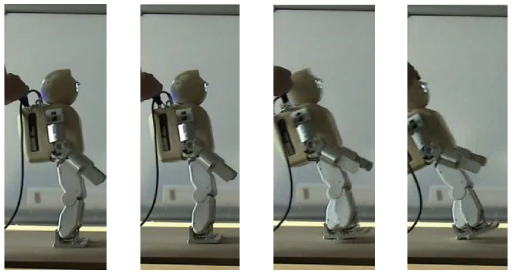


FIG. 4 – Représentation du ZMP, pour un pas optimisé en considérant 10 instants de discrétisation

$[-0,04;0,068]$, qui correspond à la taille du pied. Nous pouvons constater que les contraintes sont bien respectées pour les instants discrets considérés par l'algorithme d'optimisation. Cependant, sur toute la durée du mouvement, la contrainte du ZMP est violée, par exemple entre $t = 0.05s$ et $t = 0.1s$. Pour illustrer l'impact que cette violation de contrainte peut avoir, nous avons appliqué ce mouvement sur le robot humanoïde HOAP-3. Il s'avère donc que cette violation de contrainte entraîne la chute du robot (cf. Fig.5).



(a) $t = 0s$ (b) $t = 0.12s$ (c) $t = 0.24s$ (d) $t = 0.36s$

FIG. 5 – Mouvement optimisé en considérant 10 instants de discrétisation

Une solution pour résoudre ce problème serait d'augmenter le nombre k d'instants. Cependant cela ne permet pas d'éviter les possibles violations. De ce fait, nous présentons la contribution principale de cet article : la discrétisation garantie de contraintes.

4 Discrétisation garantie de contraintes

Le principe de la discrétisation garantie de contraintes est de calculer la valeur maximale de la fonction contrainte $g_i(\mathbf{X},t)$ pour un intervalle de temps $[t] = [\underline{t},\bar{t}]$ au lieu de la valeur calculée à un instant précis, afin de la renvoyer à l'algorithme d'optimisation. L'analyse par intervalle va nous permettre d'obtenir une implémentation de ce calcul.

4.1 Analyse par Intervalle

Initialement l'analyse par intervalle fut développée pour quantifier les erreurs d'arrondi introduites par la représentation de nombres réels par des nombres flottants dans le domaine de l'informatique. Puis elle fût étendue à la validation numérique (15; 11; 12).

Un intervalle réel $[a] = [\underline{a};\ \bar{a}]$ est un sous ensemble de $\mathbb{R}$. Avec $\underline{a} = Inf([a])$ et $\bar{a} = Sup([a])$. L'ensemble de tous les intervalles réels de $\mathbb{R}$ est noté par $\mathbb{IR}$. Les opérations arithmétiques réelles sont étendues aux intervalles. Considérons un opérateur $\circ \in \{+,-,*,\div\}$ et $[a]$ et $[b]$ deux intervalles. Alors :

$$[a] \circ [b] = [inf_{u\in[a],v\in[b]}\, u \circ v, \quad sup_{u\in[a],v\in[b]}\, u \circ v] \tag{12}$$

Considérons $\mathbf{m} : \mathbb{R}^n \longmapsto \mathbb{R}^m$; le calcul de la fonction sur un vecteur intervalle [a] est donné par :

$$\mathbf{m}([\mathbf{a}]) = \{\mathbf{m}(\mathbf{u}) \mid \mathbf{u} \in [\mathbf{a}]\} \tag{13}$$

La fonction intervalle $[\mathbf{m}] : \mathbb{IR}^n \longmapsto \mathbb{IR}^m$ est une fonction d'inclusion de $\mathbf{m}$ si

$$\forall [\mathbf{a}] \in \mathbb{IR}^n,\ \mathbf{m}([\mathbf{a}]) \subseteq [\mathbf{m}]([\mathbf{a}]) \tag{14}$$

Une fonction d'inclusion de $\mathbf{m}$ peut être obtenue en remplaçant chaque occurrence réelle par son correspondant intervalle et chaque fonction par son équivalent intervalle, dans ce cas elle est définie comme la fonction d'inclusion est appelé fonction naturelle d'inclusion. Les performances de cette fonction d'inclusion dépendent de l'expression formelle de $\mathbf{m}$.

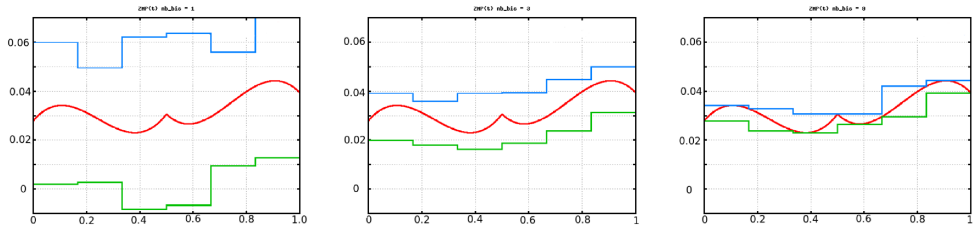
Concrètement l'analyse par intervalle offre une implémentation pratique, tout en étant un moyen garanti, de calculer les valeurs minimale et maximale d'une fonction $m(t)$ quand t est défini sur un intervalle $[t]$. Une borne *supérieure* pour la valeur maximale $\max_{t\in[t]}(m(t))$ est donnée par $Sup[m]([t])$ ainsi qu'une borne *inférieure* pour la valeur minimale $\min_{t\in[t]}(m(t))$ par $Inf[m]([t])$.

En pratique, l'encadrement peut s'avérer trop large. Cependant il existe plusieurs techniques pour obtenir un encadrement plus étroit, telles que les expansions en série de Taylor ou d'autres techniques d'optimisation globale (5).

4.2 Application

Dans cette étude nous avons utilisé la méthode la plus simple pour affiner l'encadrement et obtenir une meilleure évaluation des fonctions contraintes sur un intervalle de temps. Nous avons choisi de séparer l'intervalle initial en un ensemble de N_b sous-intervalles sur lesquels le calcul sera relancé. Le nombre de sous-intervalles dépendra de la précision désirée.

Par exemple, considérons la fonction d'inclusion $[ZMP]([t])$. Dans l'équation (5) on remarque que la variable t apparaît plusieurs fois. De ce fait l'encadrement de $[ZMP]([t])$ risque d'être très large. Grâce à la décomposition en sous intervalles, nous pouvons contrôler la précision du calcul des extremum.



(a) $N_b = 1$ (b) $N_b = 8$ (c) $N_b = 512$

FIG. 6 – Encadrement du ZMP pour 6 intervalles de temps

La Figure 6(b), montre comment la décomposition en sous intervalles agit sur la largeur de l'encadrement.

Le tableau 1 montre la corrélation entre le temps de calcul est la précision. Le temps de calcul pour un instant discret est de 0.03ms. On s'aperçoit que la discrétisation par intervalle demande plus de temps de calcul. Pour une application hors ligne qui garantit la sécurité du mouvement nous pouvons considérer que cela reste convenable. Cependant cette méthode augmente le temps de calcul et l'utilisateur doit donc faire un compromis entre la précision et le temps de calcul.

La précision est calculée de la manière suivante :

$$\text{précision}_i(\%) = \frac{\varepsilon_i - \varepsilon_{2^{15}}}{\varepsilon_{2^{15}}} * 100 \tag{15}$$

avec :ε_i = Borne_supérieure - Borne_inférieure pour un calcul fait en considérant i sous intervalles.

Nous avons choisi comme référence l'encadrement obtenus pour 2^{15} sous-intervalles.

N_b	1	2	4	8	16	32
précision(%)	72.9	48.9	19.4	8.2	3.7	1.7
temps de calcul	0.7ms	1.4ms	2.5ms	5.4ms	11ms	20ms

N_b	64	128	256	512	1024	2^{15}
précision(%)	0.8	0.34	0.17	0.08	0.04	0
temps de calcul	42ms	80ms	0.16s	0.32s	0.64s	20s

TAB. 1 – Comparaison : temps de calcul / précision

5 Planification de trajectoires garanties

5.1 Définition

La planification de trajectoires garanties se base sur les mêmes logiciels d'optimisation que les méthodes classiques de planification de trajectoires. Cependant, nous utilisons la discrétisation garantie des contraintes pour déterminer la valeur maximale des contraintes de type inégalité qui doivent être vérifiée sur tout un intervalle de temps. Par conséquent, l'équation (10) est remplacée par :

$$\forall i, \forall [t] \in \mathbf{IT} \qquad \max_{\forall \tau \in [t]} g_i(\mathbf{X}, \tau) \leq 0 \tag{16}$$

avec $\mathbf{IT} = \{[t_1], [t_2], ..., [t_{k-1}], [t_k]\}$ où $[t_n] = [t_{n-1}, t_n]$ représente un intervalle de temps

Dans ce cas, les bornes supérieurs de l'équation (16) *max* g_i peuvent être obtenues facilement en calculant la borne supérieure de la fonction d'inclusion $[g_i]$ pour un intervalle de temps $[t]$:

$$\forall i, \forall [t] \in \mathbf{IT} \qquad Sup[g]_i(\mathbf{X}, [t]) \leq 0 \tag{17}$$

5.2 Dérivée des contraintes

Nous utilisons l'algorithme d'optimisation C-FSQP (Lawrence *et al.*), qui a besoin du calcul de la dérivée des contraintes par rapport aux paramètres d'optimisation : $\frac{\partial g(\mathbf{X},t)}{\partial \mathbf{X}}$. A l'heure actuelle cette dérivée est approchée par différence finie par le logiciel C-FSQP, mais des travaux en cours permettront de le calculer par différentiation automatique.

6 Résultats

Dans cette section, nous présentons les résultats de la planification de trajectoires appliquée au robot humanoïde HOAP-3 pour un pas de 7cm minimisant la durée du mouvement. Nous utilisons le logiciel C-FSQP (Lawrence *et al.*), sur un PC (Pentium-D,3GHz).

Les résultats des processus d'optimisation sont présentés dans les tableau 2, 3 , en considérant les notations suivantes :

- k : le nombre de points ou d'intervalles pour la discrétisation,
- N_c : le nombre total de contraintes du problème d'optimisation,
- T_c : le temps de calcul,
- obj : la durée du mouvement obtenu (objectif)
- Ite : le nombre d'itérations de l'algorithme.

6.1 Discrétisation par grille

Tout d'abord nous présentons les résultats des optimisations utilisant une discrétisation par grille (Cf. Section 3.2) :

Dans le Tableau 2, le nombre k d'instants discrets considérés influe sur le temps de calcul ainsi que sur la valeur finale de l'objectif. Nous pouvons supposer que le résultat optimal est obtenu pour $k = 50$ et une durée de mouvement $obj = 0.355s$. Par conséquent, un mouvement plus rapide ($obj < 0.355$), comme les mouvements obtenus pour $k = 5, 10, 20$ provoquera la violation de certaines contraintes,

k	N_c	T_c (s)	obj	Ite
5	137	103	0.320	467
10	267	137	0.344	336
20	527	1125	0.349	1510
50	1307	1587	0.355	742

TAB. 2 – Résultats des optimisations utilisant une discrétisation par grille.

Ces calculs soulignent la chose suivante : la méthode de discrétisation par grille peut amener à la violation de contraintes pour un mouvement prétendument optimal. Dans le cas d'un robot humanoïde, la violation de contrainte peut entraîner la chute ou la destruction du robot.

6.2 Discrétisation garantie

Le Tableau 3 présente les résultats d'un planification de trajectoires garanties :

k	N_c	T_c (s)	obj	Ite
5	137	2959	0.61	688
6	163	1675	0.61	331
10	267	20676	0.48	2536
12	319	2417	0.48	254
18	475	3300	0.44	232

TAB. 3 – Résultats d'un planification de trajectoires garanties.

Le nombre d'intervalles a un effet positif sur la valeur de la fonction objectif. En effet une optimisation avec $k = 6$ retourne un mouvement de $0,61s$, alors que pour $k = 18$ la durée du mouvement obtenu est de $0,44s$. Les deux solutions produisent un mouvement qui ne violera aucune contrainte. On peut en déduire que l'optimisation faite avec $k = 18$ prend en compte des solutions rejetées quand $k = 6$. Ces solutions sont rejetées à cause de l'encadrement trop large. Par conséquent, le choix du nombre d'intervalle k est important pour la précision de l'algorithme et doit être choisi avec soin.

Le tableau 3 montre que le fait d'avoir k un multiple de 6 permet de meilleur résultat. Pour plus d'information à ce sujet, nous invitons le lecteur à lire (9).

6.3 Expérimentation

Nous avons expérimenté le mouvement d'un pas de 7cm généré par la méthode de planification de trajectoires garanties en considérant $k = 10$ sur le robot humanoïde HOAP-3. La Figure 8 montre le mouvement obtenu. On s'aperçoit que le robot conserve son équilibre contrairement au mouvement de la Figure 5. La Figure 7 montre l'évolution de la fonction ZMP. Il apparaît clairement que l' encadrement $[ZMP]$ ainsi que le fonction continue $ZMP(t)$ reste toujours dans les limites du pied :

$$-0,04 \leq ZMP(t) \leq 0,068 \tag{18}$$

La planification de trajectoires garanties assure la validité des contraintes sur toute la durée du mouvement.

7 Discussion

Cette communication présente une méthode de planification de trajectoires garanties appliquée au robot humanoïde HOAP-3. Cette méthode calcule les contraintes sur des intervalles de temps pour générer une solution optimale qui interdit toute violation de contraintes, alors que des méthodes de discrétisation classique ne donne aucune garantie, ce qui peut conduire à la chute ou à la destruction du robot.

Un inconvénient de cette méthode est le temps de calcul, trop long pour une application en ligne. Cependant, une solution serait de diminuer le nombre de sous-intervalles pour le calcul des contraintes. Mais cette

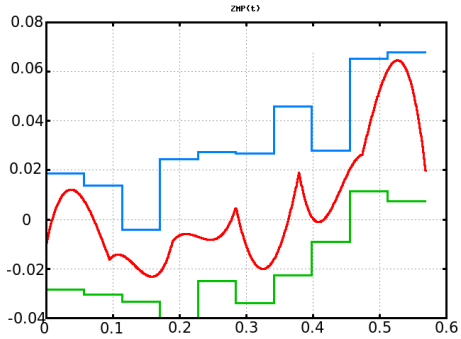
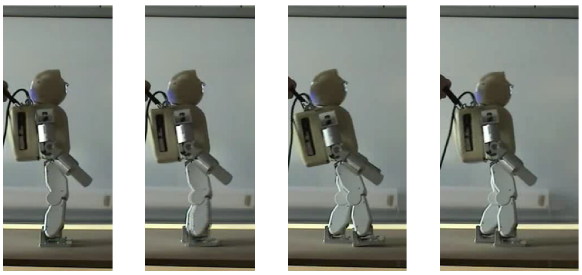


FIG. 7 – Représentation du ZMP, pour un mouvement obtenu grâce à la planification de trajectoires garanties avec $k = 10$



(a) $t = 0s$ (b) $t = 0.2s$ (c) $t = 0.4s$ (d) $t = 0.6s$

FIG. 8 – Mouvement obtenu grâce à la planification de trajectoires garanties avec $k = 10$

solution réduirait la précision du calcul, ce qui produirait des solutions sans violation mais sous-optimales. Nous prévoyons de tester d'autres solutions (5), ce qui pourrait diminuer le temps de calcul pour une précision identique. Toutefois, il est important de noter que le résultat obtenu est garanti sans violation de contraintes.

Pour permettre une réutilisation facile de ce travail nous avons créé la Guaranteed Discretization Library (GDL) disponible sur `http://www.lirmm.fr/~lengagne/GDL`. Cette librairie en C++ permet la discrétisation garantie en utilisant l'analyse par intervalle.

La méthode de discrétisation par intervalle peut être étendue à plusieurs dimensions. Dans notre cas nous n'avons considéré que la dimension temporelle. Mais une discrétisation 2-D de l'espace libre pour le planification de déplacement de robots mobiles peut facilement être envisagée. De même la discrétisation par intervalle peut s'appliquer à des espaces à N-dimensions, même si dans ce cas les temps de calculs augmenteraient considérablement.

Conclusion

Dans ce papier nous avons présenté la planification de trajectoires garanties. Cette méthode utilise des algorithmes d'optimisation classiques, mais remplace la discrétisation par grille par une discrétisation garantie des contraintes. L'évaluation expérimentale sur le robot humanoïde HOAP-3 montre clairement la nécessité d'une garantie sur les contraintes pendant les mouvements en boucle ouverte. Les mouvements, dit optimaux, obtenus par des méthodes de discrétisation par grille peuvent provoquer une violation des contraintes avec des conséquences importantes sur le robot telles que sa chute ou sa destruction. Au contraire notre méthode de planification de trajectoires garanties génère des mouvements sûrs qui ne mettent pas en danger l'intégrité et l'équilibre du robot.

Le calcul de l'encadrement des contraintes et du gradient par rapport aux paramètres nécessitent d'avantage de recherches afin de palier au problème de temps de calcul

Références

[1] DENK J. & SCHMIDT G. (2001). Synthesis of a Walking Primitive Database for a Humanoid Robot using Optimal Control Techniques. In *Proceedings of IEEE-RAS International Conference on Humanoid Robots*, p. 319–326, Tokyo, Japan.

[2] FANG H. & MERLET J. P. (2005). Dynamic interference avoidance of 2-DOF robot arms using interval analysis. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, p. 3809–3814.

[3] FRAISSE P., COTTON S., MURRAY A. & PIERROT F. (2007). Towards dynamic balance control of humanoid robots by using com and zmp. In *Workshop in the IEEE-RAS 7th International Conference on Humanoid Robots*.

[4] GUIRAUD D., POIGNET P., WIEBER P., MAKKSOUD H. E., PIERROT F., DOMBRE E., DIVOUX J. & RABISCHONG P. (2003). Modelling of the human paralysed lower limb under fes. In *Robotics and Automation, 2003. Proceedings. ICRA apos ;03. IEEE International Conference on*, volume 2, p. 2218–2223.

[5] HANSEN E. & WALSTER G. (2004). *Global optimization using interval analysis*. Marcel Dekker, 2nd edition.

[6] HETTICH R. & KORTANEK K. O. (1993). Semi-infinite programming : theory, methods, and applications. *SIAM Rev.*, **35**(3), 380–429.

[7] KHALIL W. & DOMBRE E. (2002). *Modeling, Identification & Control of Robots*. Hermes Sciences Europe, 3 edition.

[Lawrence *et al.*] LAWRENCE C., ZHOU J. L. & TITS A. L. *User's Guide for CFSQP Version 2.5 : A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints*. Electrical Engineering Department, Institute for Systems Research University of Maryland, College Park, MD 20742.

[8] LENGAGNE S., RAMDANI N. & FRAISSE P. (2007). Guaranteed computation of constraints for safe path planning. In *IEEE-RAS 7th International Conference on Humanoid Robots*.

[9] LENGAGNE S., RAMDANI N. & FRAISSE P. (2008.). Safe path planning for humanoid robots. In *submitted to Intelligent Robots and Systems, (IROS 2008)*.

[10] MIOSSEC S., YOKOI K. & KHEDDAR A. (2006). Development of a software for motion optimization of robots - application to the kick motion of the hrp-2 robot. In *Proceedings of the 2006 IEEE International Conference on Robotics and Biomimetics*, p. 299–304.

[11] MOORE R. E. & BIERBAUM F. (1979). *Methods and Applications of Interval Analysis (SIAM Studies in Applied and Numerical Mathematics) (Siam Studies in Applied Mathematics, 2.)*. Soc for Industrial & Applied Math.

[12] NEUMAIER A. (1990). *Interval methods for systems of equations*. Cambridge : Cambridge university press.

[13] PIAZZI A. & VISIOLI A. (2000). Global minimum-jerk trajectory planning of robot manipulators. In *IEEE Transactions on Industrial Electronics*, volume 47, p. 140–149.

[14] REEMTSEN R. (1998). Semi-infinite programming : discretization methods.

[15] SUNAGA T. (1958). Theory of interval algebra and its application to numerical analysis. *RAAG Memoirs, Ggujutsu Bunken Fukuy-kai*, **2**, 547–564.

[16] VON STRYK O. (1993). Numerical solution of optimal control problems by direct collocation.

[17] VUKOBRATOVIC M. & JURICIC D. (1969). Contribution to the synthesis of biped gait. *IEEE trans. Bio-Med Eng.*, **BME-16**, 1–6.

[18] YOSHIDA E., BELOUSOV I., ESTEVES C. & LAUMOND J.-P. (2005). Humanoid motion planning for dynamic tasks. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, p. 1–6.

Processus décisionnels de Markov décentralisés multi-critères pour la formation de peloton

Matthieu Boussard[1], Maroua Bouzid[1], Abdel-Illah Mouaddib[1]

Université de Caen, Laboratoire GREYC UMR 6072, Campus Côte de Nacre, bd du Maréchal Juin, 14032 Caen Cedex ,France
{ matthieu.boussard, bouzid, mouaddib }@info.unicaen.fr

Résumé : Cet article traite du problème de l'organisation d'un groupe d'agents mobiles de sorte de les faire parvenir d'une zone de départ à une zone d'arrivée le plus rapidement possible. La formation de peloton[1] est une manière efficace d'organiser ces agents. Un peloton est un groupe d'agents où chaque agent doit se coordonner avec ses proches voisons. Ainsi ils doivent adapter leurs mouvements afin de gérer la distance inter-agent, ainsi que les changements de l'environnement. Dans de nombreuses approches un leader doit être désigné afin de gérer le peloton. Dans cet article, nous proposons une approche totalement décentralisée, où aucun leader n'est désigné à priori ; non seulement, il émerge à partir des comportements locaux, mais de plus il est capable de changer selon la situation. Tout d'abord, nous formaliserons le problème de platooning (formation et déplacement du peloton) par un 2V-DEC-MDP (Vector-Valued Decentralized Markov Decision Process), puis nous proposerons un algorithme afin de le résoudre, nous présenterons ensuite des résultats sur des exemples concrets ainsi que des propriétés de l'approche illustrant sa robustesse et sa pertinence pour son utilisation dans des environnements dynamiques.

1 Introduction

La nécessité de coordonner les déplacements d'agents situés est crucial dans de nombreuses applications réelles, comme en particulier lorsqu'un groupe d'agents doit se déplacer d'un endroit à un autre. De nombreuses approches sont dédiées à ces problèmes :

- les techniques de flocking où le comportement des agents y est décrit par trois règles simples : (1) séparation, éviter de bousculer ses voisins, (2) alignement, se diriger dans la direction moyenne de ses voisins, (3) cohésion, se diriger dans la position moyenne de ses voisins,
- la répartition spatiale (foraging robots), où le but des agents est de couvrir la plus grande zone possible,
- le déplacement ou la formation en peloton, où les agents doivent de déplacer afin suivre un leader.

La propriété commune de tout ces mouvements de groupe est que le comportement global émerge de comportements locaux. Notre problème général est de formaliser ce comportement en processus décisionnel et ses interactions locales, afin d'atteindre un comportement global cohérent. Ce type d'application est très utile dans certaines applications robotiques réelles précises tel le déminage, le déplacement en convois, l'exploration, la recherche et le sauvetage de civils.

Dans cet article nous nous intéressons plus particulièrement à la formation de pelotons. Dans ce contexte, plusieurs approches se sont fondées sur des techniques de control longitudinal (Naffin *et al.*, 2004). L'utilisation de perceptions locales permet aux agents de maintenir une certaine distance aux agents les plus proches. Dans les systèmes utilisant les techniques de contrôle longitudinal, il est nécessaire d'avoir un leader dans le peloton permettant au groupe de se diriger vers le but. Ces systèmes ne prennent pas en compte les incertitudes de manière explicites, ni d'exprimer différents types d'interactions entre agents, tel des contraintes de précédence entre les actions (Beynier & Mouaddib, 2006), ou bien encore le partage de ressources communes.

Dans (Mouaddib *et al.*, 2007), un cadre formel a été développé afin de formaliser les impacts d'une décision locale sur le groupe en considérant deux nouveaux critères, basé respectivement sur les effets positifs et négatifs. Le processus de décision local de chaque agents considère l'intérêt personnel d'un agent et l'intérêt du groupe. Un cadre formel fondé sur les Vector-Value MDP (Kazuyoshi Wakuta, 2000), où la

[1] *Platoon* en anglais

fonction de valeur espérée est exprimée par un vecteur dont chaque composante représente l'espérance sur un critère. Cette approche considère toutes les interactions dans tous les états ce qui conduit à une complexité trop importante. Afin de réduire cette complexité, dans (Boussard *et al.*, 2007) seul les interactions locales ont été prises en compte. Il s'agit d'une approche basée sur des optimisations locales permettant aux agents d'évaluer l'impact de ses décisions uniquement sur les agents qu'il perçoit. Dans cette approche, un agent perçoit un nombre restreint d'agents dans son environnement, construit en ligne un 2V-MDP afin de calculer une politique permettant à l'agent de se comporter dans son voisinage en prenant en compte les agents perçus. L'algorithme proposé satisfait les contraintes suivantes :

- **Changement dynamique** : La complexité pour un agent de calculer sa politique est celle d'un MDP, cela permet en cas de changement dans l'environnement, de la recalculer rapidement.
- **Passage a l'échelle** : Notre approche est applicable dans un système réel car chaque agent construit un petit 2V-MDP avec un nombre restreint d'états, car le problème de décision est formalisé pour un environnement limité. De plus, le DEC-MDP est considéré comme un ensemble de MDPs.
- **Coordination locale** : Notre approche se base sur des perceptions locales. Cela implique qu'un agent ne peut interagir qu'avec un nombre limité d'agents, en un nombre limité d'états. Cela réduit significativement la complexité du problème.
- **Optimal si possible** : Le comportement d'un agent doit être optimal lorsqu'il se trouve dans une situation "facile".

Dans cet article, nous montrons comment ce formalisme peut-être utilisé afin de résoudre le problème de platooning. Dans la section 2, nous présentons les travaux connexes et les fondements nécessaires pour traiter de ce problème. La section 3 présentera de manière plus formelle le problème du platooning. Section 4 présente un cadre formel basé sur les 2V-DEC-MDP (Mouaddib *et al.*, 2007; Boussard *et al.*, 2007) pour résoudre ce problème. Section 5 les outils pour résoudre le 2V-DEC-MDP obtenu. Section 6 montre les premiers résultats obtenus. Section 7 aborde quelques problèmes théoriques ouvert, et nous conclurons cet article Section 8.

2 Travaux connexes

Le problème traité dans cet article peut être vu comme un problème d'intelligence collective et de planification multi-agent (Wolpert & Tumer, 2002), ce qui a attiré l'attention de nombreux chercheurs en intelligence artificielle. Il est possible de considérer le platooning comme une forme de flocking (Lindhé *et al.*, 2004), ou chaque agent doit maintenir la cohésion du groupe. Des formalismes ont donc été proposé afin de résoudre ces problèmes (Reynolds, 1999). Le principe est de maintenir une forme globale du groupe d'agents, chaque agent ne percevant seulement que leur environnement local et leurs plus proches voisins. Le plus gros intérêt de ces méthodes est leur capacité à gérer un très grand nombre d'agents (grâce aux contraintes locales). Néanmoins, ces approches souffres d'un manque de preuve d'optimalité ainsi que l'une faible expressivité dans la prise en compte de différents types d'interactions.

Le contrôle longitudinal est utilisé dans les problème de platooning (Naffin *et al.*, 2004; Simon Hallé and Brahim Chaib-draa, 2005), comme dans le cas des applications d'adaptative cruise control. Ces techniques ont pour but de respecter une distance de sécurité entre les véhicules, par coordination locale. Le leader du peloton peut aussi donner des ordres au reste du groupe, comme pour permettre à un agent de s'insérer dans le peloton. Dans ces approches, le peloton doit avoir un leader qui est le seul membre du groupe à avoir un but. Néanmoins, le grand nombre de messages échangés entre les membre du peloton peut être un problème dans certaines applications. Les techniques de planification permettent une description précise de l'environnement ainsi que du but recherché. Elles peuvent être divisées en deux classes :

- centralisés, où un agent unique calcule le comportement du groupe,
- décentralisés, ou chaque agent choisi l'action à exécuter.

Le principal avantage des approches centralisées est la preuve d'optimalité du comportement des agents, l'inconvenant étant l'apparition d'un goulot d'étranglement, ainsi que d'une explosion combinatoire comme dans les Multi-agent Markov decision Processes (MMDP) (Boutilier, 1996). Les DEC-MDP (Bernstein *et al.*, 2000) ont été proposé afin de permettre de traiter des application décentralisés, mais leur complexité dans le cas général est trop importante pour leur utilisation dans des applications réelles. Néanmoins, de récents travaux ont montré la possibilité d'utiliser cette formalisation pour des applications de grande taille (Beynier & Mouaddib, 2006) en considérant certaines interactions spécifiques, ainsi que des interactions locales. Notre approche est similaire, dans son concept d'interaction locales, mais offre un modèle d'inter-

action plus riche.

3 Le platooning

Soit un groupe d'agents dans une zone de départ et qui doivent rejoindre une zone but le plus rapidement possible. Il n'y a pas d'exigence sur l'ordre d'arrivé. La securité des agents est un des point cruciaux. A cause des incertitudes sur le résultat des actions, les agents doivent respecter une distance de sécurité. Si cette distance est trop importante, cela peut nuire à l'efficacité de la solution. Ainsi nous définissions un peloton d'un groupe agent comme étant un ensemble de robots mobiles, chacun cherchant à gérer sa distance à ses voisins, de sorte que le peloton puisse rejoindre le but efficacement. Figure 1 decrit un problème de peloton.

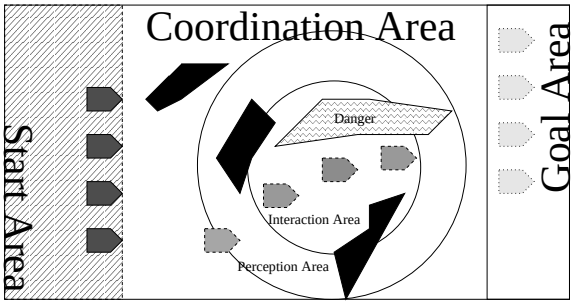


FIG. 1 – Un peloton

Nous supposons que les agents évoluent dans un environnement discret (dans une application réelle, une phase de discrétisation de l'environnement est nécessaire), totalement observable sans les autres agents, ceux-ci ne seront perçus que dans un voisinage limité. Tous les agents utilisent le même modèle du monde et, pour simplifier, connaissent tous le même but. Il peut subvenir des changements dans l'environnement. Nous supposons de plus que les agents disposent d'un nombre fini d'actions. Nous supposons enfin les actions comme étant stochastiques, les résultats des actions étant soumis à une distribution de probabilité. Dû aux communications limitées (voir inexistantes), nous considérons que les agents ne communiquent pas pour se coordonner. Nous ne désignons pas non plus de leader pour le peloton, chaque agent pouvant, selon la situation, prendre la direction du peloton.

4 Formalisation

Nous décrivons comment exprimer le problème de platooning comme un 2V-DEC-MDP. Ce modèle offre une description précise du problème de planification multi-agent. Il exhibe explicitement toutes les actions possibles par agents, les incertitudes sur le résultat de celle-ci, ainsi que le but des agents. Ceci permet une coordination précise. En effet, si deux agents sélectionnent deux actions quasiment déterministes, il y a un très faible risque de collision, par opposition à deux actions très incertaines.

Soit un ensemble $\mathcal{A}$ représentant une flotte d'agents. L'environnement dans lequel ils évoluent, représenté par un ensemble d'états S, est considéré comme étant totalement observable (l'état du monde en ne considérant pas les autres agents) ainsi l'état observé $s \in S$ est le vrai état du système ($P(s|Obs(s)) = 1$). Cette hypothèse pourrait être relaxée, mais le modèle deviendrait un 2V-DEC-POMDP dans lequel nous ne pourrions trouver un algorithme efficace. Chaque agent dispose de son ensemble d'actions A_i. La fonction de transition $P(s, a, s', \forall s, s' \in S, a \in A_i$ permet d'exprimer l'incertitude sur le résultat des actions. Le but est représenté par un ensemble d'état but $G \subset S$, where $R(s) = v, \forall s \in G$. Chaque agent partagent les même :

- ensemble d'états,
- fonction de récompense. Celle-ci permet de représenter à la fois les récompense pour les état but, mais aussi, par des récompenses négatives, les états dangereux.

Chaque agent dispose par ailleurs de son propre :

- ensemble d'actions,

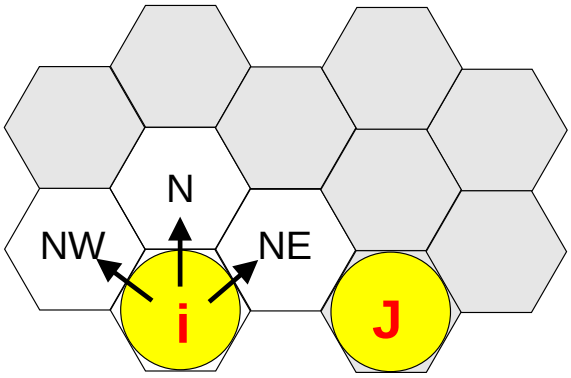


FIG. 2 – Calcul de l'ensemble D_i

– fonction de transition $P(s, a, s'), s, s' \in S, a \in A_i$.

4.1 Le formalisme 2V-DEC-MDP

Le monde dans lequel évolue les agents est représenté par un MDP $\langle S, A, T, R\rangle$. Un état $s \in S$ est constitué d'un ensemble d'observations sur le monde. Néanmoins il ne s'agit pas d'observabilité partielle car ces observations représentent l'état réel de l'agent.

Définition 1
Pour un agent i et une action $a \in A_i$, les préconditions nécessaires à l'exécution d'une action sont notées : : $in(a, s) : A \times S \rightarrow \{true, false\}$

Définition 2
Etant donné un état et une action, l'ensemble des états résultants (à cause de l'incertitude) est donné par : $out(a, s) : A \times s \rightarrow s'$

Quand un agent doit adapter son comportement à celui du groupe, il doit calculer l'impact de ses actions sur son voisinage. Il y a de nombreux types d'interactions possibles : physiques (collisions, obstructions), mais aussi

$$\begin{aligned}
E_i(s, a) &= \{b \in A_{j\neq i} | in(a) \subset s \wedge in(b) \nsubseteq s \\
&\quad \wedge \exists s' \in out(a, s) : in(b) \subset s'\} \\
D_i(s, a) &= \{b \in A_{j\neq i} | in(a) \subset s \wedge in(b) \subseteq s \\
&\quad \wedge \exists s' \in out(a, s') : in(b) \nsubseteq s'\} \\
Eb_i(s, a) &= \{b \in A_{j\neq i} | in(a) \nsubseteq s \wedge in(b) \subset s \\
&\quad \wedge \exists s' \in out(b, s) : in(a) \subset s'\} \\
Db_i(s, a) &= \{b \in A_{j\neq i} | in(a) \subset s \wedge in(b) \in s \\
&\quad \wedge \exists s' \in out(b, s) : in(a) \nsubseteq s'\}
\end{aligned}$$

Dans le problème de platooning, seules quelques-unes de ces relations sont importante. La relation D_i décrit les actions que l'agent va bloquer chez d'autre agents. Cela permet d'exprimer des obstructions. C'est cette relation que nous utiliserons pour gérer les interactions dans le peloton.

4.2 Le platooning modélisé par un 2V-DEC-MDP

Afin d'illustrer l'idée de la formalisation, nous considérons l'exemple Figure 2 où une situation d'interaction est présentée. Soit i et j deux agents. Leur ensemble d'actions est $\{N, NE, SE, S, SW, NW\}$. Chaque hexagone représente un état. En ne considérant pas les autres agents, le monde est totalement observable. Les agents sont perçus seulement localement. Ainsi l'état s contient la position exacte de l'agent i ainsi que la position exacte de ses voisins. Nous décrivons ici les différentes étapes pour le calcul de D_i (ensemble *Disable* pour l'agent i).

– Sélection d'une action
  $in(NE) \subset s$
– Pour toutes les actions possibles chez un autre agent
  $b \in A_{j \neq i}, in(b) \subseteq s = \{NW, N, NE\}$
– Calcule de D_i
  $D_i = \{NW\}$

Une fois cet ensemble d'interactions déterminés, l'agent peut se représenter l'impact de son action sur le groupe. À chaque action possible a dans un état s, un vecteur de valeur $(ER(s,a), JER(s,a), JEPenalty(s,a))$ est attribué. Ce vecteur représente respectivement, la récompense espérée (ER), le gain espéré pour le groupe (JER), le coût occasionné ($JPenalty$). Les ensembles E_i, D_i, Eb_i et Db_i sont utilisés pour établir les valeur de ces fonctions. Plus de détail sont donnés dans (Boussard *et al.*, 2007).

Pour le problème de platooning, nous définissions les fonctions $(ER(s,a), JER(s,a), JEPenalty(s,a))$:

$$ER(s,a) = R(s,a) + \sum_{s' \in S} p(s,a,s')V^*(s') \quad (1)$$

$$JER(s,a) = 0 \quad (2)$$

$$JEPenalty(s,a) = \sum_{s' \in S} p(s,a,s') \sum_{s_j,a_j} p(s_j,a_j,s')C \quad (3)$$

Il est nécessaire d'obtenir la probabilité de collision pour calculer $JEPenality$, D_i n'apparaît pas dans l'équation 3, mais $\sum_{s' \in S} p(s,a,s') \sum_{s_j,a_j} p(s_j,a_j,s')$ à la même signification (le calcul est ajusté avec les probabilités des actions). C est un coût de collision permettant d'introduire des priorités entre les agents (par défaut,$C = 1$). Le critère ER représente l'utilité espérée, en utilisant la fonction de valeur de la politique mono-agent optimale (Equation 1). Comme le but du groupe est d'atteindre le but le plus rapidement possible un agent se rapprochant du but augment l'utilité du groupe, c'est pourquoi nous avons définit $JER = 0$ afin de ne pas être redondant avec ER. Afin de trouver la politique optimale, il est nécessaire d'optimiser les trois équations ER, JER et $JEPenalty$ comme présenté dans (Mouaddib *et al.*, 2007).

5 Une technique basée sur le regret pour la résolution de 2V-DEC-MDP

Les processus décisionnel de Markov décentralisés à valuation vectorielle (2V-DEC-MDP) est une extension des MDP à valuation vectorielle, qui permettent une formalisation des interaction entre les agents, et offrent un processus décisionnel réalisant un compromis entre l'intérêt personnel et l'intérêt du groupe (en évaluant les fonctions d'interactions). L'algorithme est constitué de deux parties :

– la première partie consiste pour tous les agents en le calcul des leur fonction de valeurs optimale V^*, donné par le MDP $\langle S, A, P, R \rangle$. Pour tout $s \in S$, V^* est donné par :

$$V^*(s) = R(s) + \gamma \max_{a \in A_t} \sum_{s'} p_t(s,a,s')V^*(s')$$

  Mais cela ne permet pas au groupe d'atteindre le but de manière efficace. C'est pourquoi la seconde partie de l'algorithme va adapter la décision de l'agent étant donné les agents dans la zone d'interaction.
– Avant chaque décision, tous les agents calculent les impacts de chacune de leur actions, construisent un 2V-MDP $\langle S, A, P, R, JEPenalty \rangle$. Résoudre ce 2V-MDP est un problème d'optimisation multi-critère, où l'agent doit choisir l'action maximisant le vecteur :

$$(R(s,a) + \gamma \sum_{s'} p(s,a,s')V^*(s'), JEPenalty(s,a))$$

L'algorithme basé sur le regret (Boussard *et al.*, 2007) pour résoudre de tel problème consiste en deux étapes :

– premièrement, l'utilisation d'optimisations mono-critère pour chaque critère (en résolvant le MDP à l'aide, par exemple, de Value Iteration) permet d'obtenir un vecteur de valeurs idéales pour chaque critère.

– Pour chaque action, un vecteur de regret est déterminé en calculant la distance entre les valeurs de chaque action, et la valeur idéale. Ensuite, l'action est choisie selon un ordre Leximin (lexicographiquement, du plus petit regret au plus important).

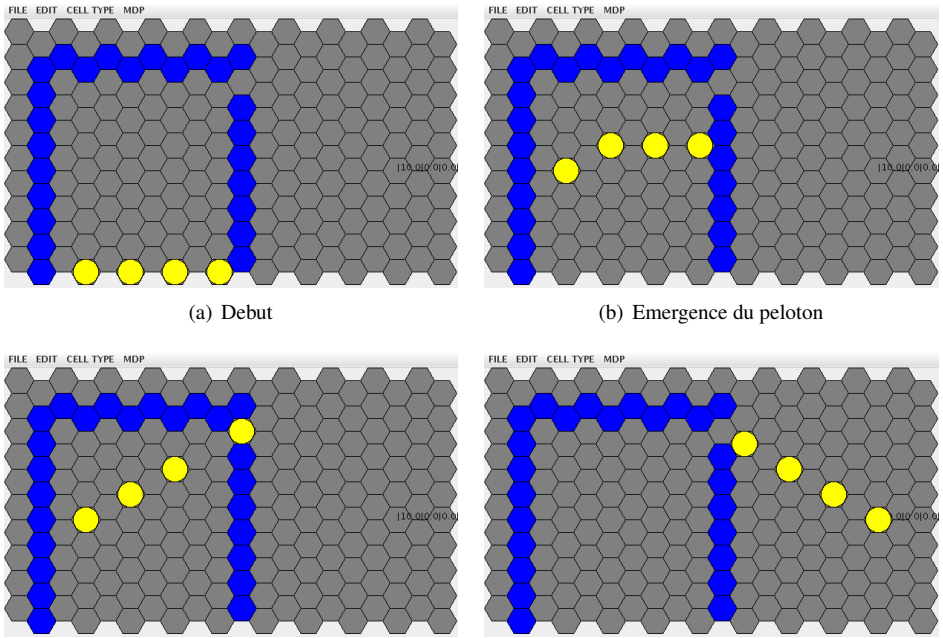
Cet algorithme ne garantit pas le critère d'optimalité de Bellman, néanmoins il a montré apporter des solutions satisfaisantes en choisissant des actions réalisant de bon compromis.

6 Premières évaluations

Ces résultats, obtenus par simulation, illustrent le comportement du peloton dans deux situations. Nous présenterons aussi des résultats de complexité.

6.1 Emergence du peloton

Figure 3 montre la formation émergente du peloton. Dans la Figure 3,l'état de départ est la position initiale des robots. Figure 3b, l'agent le plus à gauche ne peut plus avancer sans faire un détour, et ne peut se rapprocher vers la droite à cause du risque de collision. L'agent préfère s'arrêter et attendre de voir si son voisin bouge. C'est le début de la formation de peloton. Figure 3c, tous les agents ont raisonné de la même manière, jusqu'à ce que l'agent le plus à droite puisse sortir. L'agent qui le suivait peut maintenant se déplacer sans risque de collision. Figure 3d, le peloton s'est formé et sort de la zone de départ. Cet exemple a montre l'émergence d'un agent dans le rôle du leader, bien qu'il n'ait pas été désigné au préalable. Cette émergence du leader peut être atteinte dans des situations plus complexes.



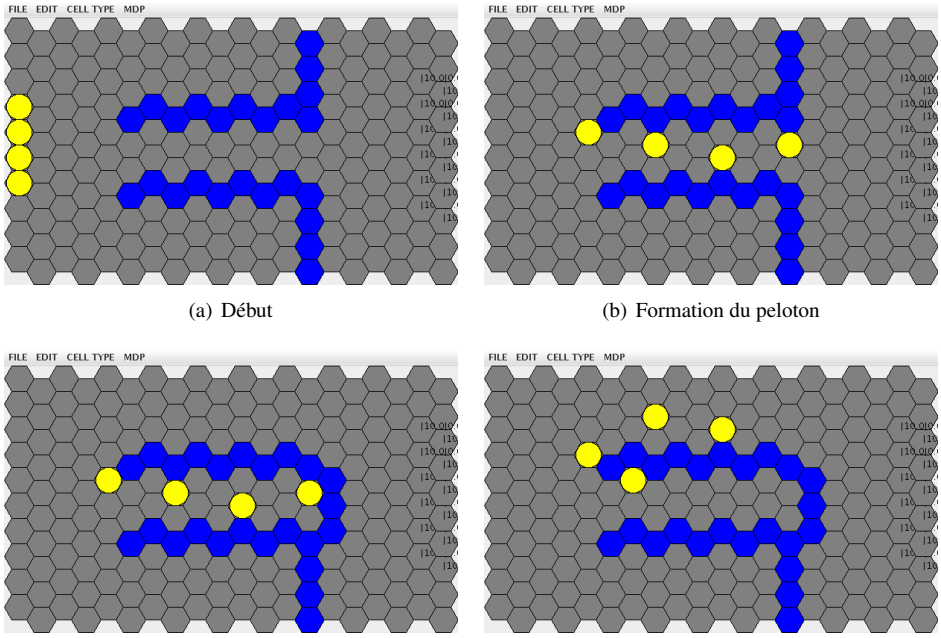
(a) Debut (b) Emergence du peloton

(c) Déplacement du leader (d) Déplacement du peloton

FIG. 3 – Emergent platoon

6.2 Changement de leader

Figure 4 montre comment le leader du peloton peut changer au cours d'une mission. Nous présentons un peloton se déplaçant vers un but a travers un tunnel lorsque celui-ci se bouche. Figure 4a est l'état de départ, de là les agents doivent passer entre les murs. Figure 4b, le peloton est formé, il commence à avancer. Figure 4c, l'agent découvre que la sortie est bouchée, les agents calculent alors un autre chemin pour atteindre le but. Figure 4d, le peloton sort alors du tunnel et se dirige vers le but.



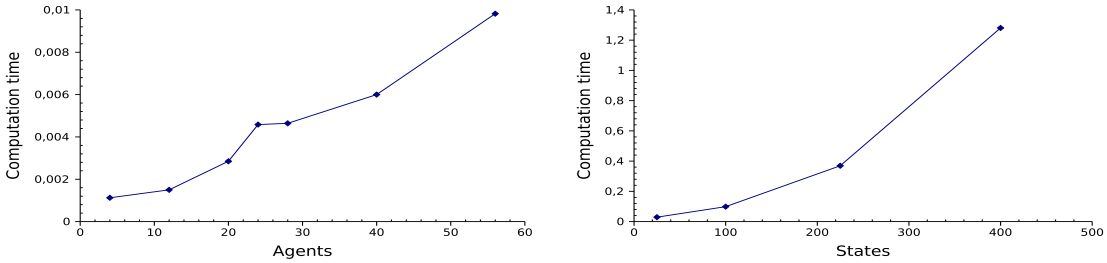
(a) Début (b) Formation du peloton

(c) Replanification, nouveau leader (d) Déplacement du peloton

FIG. 4 – Emergence d'un nouveau leader

6.3 Complexité et performances

Figure 5 montre les performances de l'algorithme. À chaque étape l'agent doit calculer pour toutes ses actions, l'impact sur tous les agents perçus. Cette partie en ligne (l'adaptation aux autres agents) est linéaire en le nombre de voisins (Figure 5a). Pour 50 agents, il faut 10ms pour choisir une action, ce qui semble réaliste pour une application embarquée réelle. Ces expérimentations supportent nos arguments sur la capacité de notre approche à gérer un grand nombre d'agents (nous avons gérer 60 agents) , ainsi que de réagir aux changement de l'environnement (la complexité pour résoudre un MDP pour un unique agent étant acceptable Figure 5b).



(a) Complexité en ligne (b) Planification

FIG. 5 – Complexité et performances

7 Discussion et problèmes théoriques

Nous souhaitons mettre en avant des problèmes théoriques ouverts soulevés par le formalisme des 2V-DEC-MDP sur les problèmes de platooning.

7.1 Emergence du leader

Cette approche ne désigne jamais un leader explicitement. Il apparaît néanmoins que ce leader existe bien. Une fois le peloton formé, si le leader s'arrête, tout le peloton s'arrêtera aussi. Dans la Figure 3, il

n'y a pas de leader tant que le peloton n'est pas formé et qu'il commence a bouger (potentiellement un autre agent aurait pu prendre le rôle du leader). L'analyse de l'émergence du leader est un des plus gros problème théorique de cette approche. Il est constitué de deux parties : Comment repérer le leader dans un groupe d'agents ? Une fois identifié, nous pourrions utiliser des techniques comme dans (Könönen, 2003) pour améliorer le comportement global. De plus les cas de changement de leader sont à étudier aussi.

7.2 L'impact à long terme des actions

Les 2V-DEC-MDP permettent d'exprimer l'impact à court terme des actions. Mais il apparaît que quelques actions d'un petit groupe agents (d'un seul ?) affectent le groupe tout entier. Par exemple si un agent bloque une porte, le reste du groupe ne pourra plus passer. En libérant le passage, tout le groupe pourra alors potentielement passer.

À la fin, ou bien en cas d'inter-blocage, le peloton fini par tomber dans un état d'équilibre. L'étude de cet équilibre fait parti des travaux actuels. Nous nous intéressons aux trois types d'équilibres que sont Nash (absence de regret), Pareto (ne peux mieux faire sans causer de dommages) et Stackelberg (présence d'un leader). D'une part, comme les agents sélectionnent des actions qui réalisent des choix équilibrés entre l'intérêt personnel et l'intérêt du groupe, Pareto peut être un équilibre intéressant. D'autre part, si le leader agit réellement comme tel, les agents devraient aboutir dans un équilibre de Stackelberg.

8 Conclusion

Un formalisme basé sur des observations locales a été présenté dans cet article. Il utilise les MDP pour décrire le problème de planification, et exprime les interactions locales dans un 2V-DEC-MDP. Cela a permis son utilisation dans le problème de platonning, en faisant émerger un peloton sans désignation préalable d'un leader. De plus en cas de nécéssité, un nouveau leader peut émerger. Néanmoins il peut arriver que les agents tombent dans un équilibre avant d'arriver au but, cet équilibre est provoqué par un petit nombre d'agents. L'utilisation d'algorithme d'apprentissage par renforcement (Chapelle *et al.*, 2002) pour briser ces équilibres est une des extensions de ce travail. La seconde partie du travail se portera sur l'étude des équilibres, et cela dans le but de sélectionner l'agent qui doit apprendre afin de débloquer le peloton.

Références

BERNSTEIN D. S., ZILBERSTEIN S. & IMMERMAN N. (2000). The complexity of decentralized control of markov decision processes. In *UAI '00 : Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, p. 32–37 : Morgan Kaufmann Publishers Inc.

BEYNIER A. & MOUADDIB A. (2006). An iterative algorithm for solving constrained decentralized markov decision processes. In *AAAI*.

BOUSSARD M., BOUZID M. & MOUADDIB A.-I. (2007). Multi-criteria decision making for local coordination in multi-agent systems. In *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'07)*.

BOUTILIER C. (1996). Planning, learning and coordination in multiagent decision processes. In *TARK '96 : Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, p. 195–210 : Morgan Kaufmann Publishers Inc.

CHAPELLE J., SIMONIN O. & FERBER J. (2002). How situated agents can learn to cooperate by monitoring their neighbors' satisfaction. In *ECAI*, p. 68–72.

KAZUYOSHI WAKUTA (2000). *Vector valued Markov Decision Processes with Average Rewad Criterion*, volume 14, chapter Journal of Probability in the Engineering and Information Sciences, p. 533–548. Cambridge.

KÖNÖNEN V. (2003). Asymmetric multiagent reinforcement learning. In *IAT '03 : Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, p. 336, Washington, DC, USA : IEEE Computer Society.

LINDHÉ M., ÖGREN P. & JOHANSSON K. H. (2004). Flocking with obstacle avoidance : A new distributed coordination algorithm based on voronoi partitions. In *Proceedings IEEE Conference on Robotics and Automation*.

MOUADDIB A., BOUSSARD M. & BOUZID M. (2007). Towards a framework for multi-objective multiagent planning. In *AAMAS*.

NAFFIN D. J., SUKHATME G. S. & AKAR M. (2004). Lateral and longitudinal stability for decentralized formation control. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*, p. 421–430.

PUTERMAN M. L. (1994). Markov decision processes : Discrete stochastic dynamic programming. In *John Wiley and Sons, New York, NY*.

REYNOLDS C. W. (1999). Steering behaviors for autonomous characters. In *Proceedings Game Group Conference on Game developers*, p. 763–782.

SIMON HALLÉ AND BRAHIM CHAIB-DRAA (2005). *Applications of Agent Technology in Traffic and Transportation*, volume 8 of *Whitestein Series in Software Agent Technologies*, chapter Collaborative Driving System Using Teamwork for Platoon Formations. Birkhäuser.

WOLPERT D. H. & TUMER K. (2002). Collective intelligence, data routing and braess' paradox. In *JAIR*.

Une solution pour le problème du contrôle d'un troupeau de moutons avec un modèle DEC-MDP

Jalel Boughizane, Narjes Doggaz, Mourad Saadana

FACULTE DES SCIENCES DE TUNIS jalelboug@yahoo.com, Narjes.doggaz@fst.rnu.tn et Mourad.saadana@yahoo.fr

Résumé : Cet article s'inscrit dans les domaines de l'apprentissage par renforcement et Système multi-agents. Il traite du problème du contrôle d'un troupeau de moutons. Ce problème fait partie de la famille des problèmes décisionnels de grandes tailles et est modélisé par les processus de décision markoviens multi-agents et plus précisément les processus de décision markoviens décentralisés (DEC-MDP). Pour résoudre ce type de problème, nous avons définit deux méthodes de réductions d'états à savoir la décomposition et l'agrégation.

Mots-clés : Apprentissage par renforcement, DEC-MDP, Décomposition, Agrégation.

1 Introduction

L'apprentissage par renforcement est l'une des méthodes de traitement des problèmes décisionnels. Il fait appel aux processus décisionnels markoviens (MDP) pour modéliser le problème afin de produire une stratégie de résolution. Il reste cependant difficile de traiter dans le cadre MDP des problèmes réalistes dont l'espace d'état est de grande taille. Cucala a définit un problème DEC-MDP de grande taille par un problème dont les espaces d'états et d'actions sont multidimensionnels et de nombre d'états supérieur à 10^6 (Cucala et al, 2002) .

Obtenir une solution optimale pour des problèmes de grande taille apparaît peu réaliste. Par ailleurs, l'optimalité est un critère qui peut être moins important qu'il n'y paraît. Dans de nombreux problèmes, on souhaite avant tout obtenir une solution de bonne qualité. L'optimalité passe au second plan. Bien entendu, s'il est possible d'obtenir la solution optimale, cette dernière est préférée. Cependant, quand l'obtention d'une telle solution demande des ressources (en temps, en mémoire) non envisageables, une solution presque optimale sera préférée à pas de solution du tout. Dans de telles situations, on cherche à obtenir la meilleure solution possible en fonction du temps et des ressources dont nous disposons.

L'alternative est alors de décomposer ces problèmes en sous problèmes relativement indépendants et de réduire le nombre d'états en les regroupant par des techniques d'agrégations. Par ailleurs, l'utilisation de plusieurs agents s'avère utile. En effet, elle permet de réduire la complexité du problème. L'idée serait alors de combiner l'approche multi-agents et l'apprentissage par renforcement pour résoudre le problème du contrôle d'un troupeau de moutons. Cette combinaison augmenterait probablement l'éfficacité des agents dans la résolution du problème.

2 Motivation : Contrôle d'un troupeau de moutons

Le problème du contrôle d'un troupeau de moutons (CTM) est inspiré de la version de Vaughan (Vaughan *et al.*, 2000), qui consiste à faire apprendre à un robot réel comment diriger un troupeau d'oies domestiques.

Le problème du contrôle d'un troupeau de moutons (Flacher, 2005) est une expérience dans laquelle un groupe d'agents (bergers) doit conduire un groupe d'agents (moutons) vers une zone précise au sein de l'environnement (enclos). L'environnement est composé de trois bergers et six moutons (figure 1). Ces derniers ont une vitesse identique à celle des bergers et admettent un champ visuel limité. Tous les agents évoluent dans une arène circulaire. Le comportement des moutons résulte de la combinaison de trois tendances à savoir (Flacher, 2005) :

- Ils évitent de trop se rapprocher des murs qui bordent l'arène.
- Ils cherchent à se rapprocher des autres moutons présents dans les limites de leur champ visuel.

– Ils cherchent à fuir des bergers présents dans les limites de leur champ visuel.

Le but des bergers est atteint lorsque tous les moutons sont dans une zone désignée comme but à savoir l'enclos.

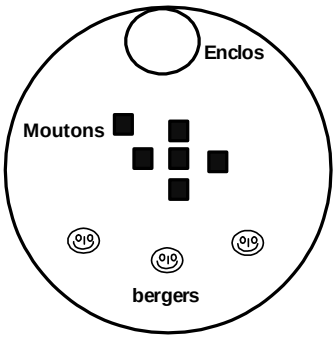


FIG. 1 – Environnement

L'approche utilisée par F. Flacher pour traiter des problèmes de coordination spatiale comme le problème contrôle d'un troupeau de moutons, dans des univers continus et dynamiques, est une approche ascendante (Flacher, 2005) fondée sur la combinaison de forces d'attraction et de répulsion entre tous les agents et les objets pertinents dans leur environnement. Dans l'approche ascendante, ces forces sont définies à un niveau élémentaire puis modifiées localement jusqu'à atteindre le comportement global recherché. F. Flacher implémenté un algorithme génétique pour trouver automatiquement des forces qui permettent aux agents d'atteindre un objectif de coordination qui s'exprime comme l'optimisation d'un ensemble de critères numériques de performance.

Le but de notre travail est de résoudre le problème de CTM à l'aide d'un modèle DEC-MDP.

3 Processus de Décision Markoviens Décentralisés(DEC-MDP)

Un DEC-MDP est un tuple $< S, A_i, P, \Omega_i, O, R>$ (Bernstein *et al.*, 2002)

- S correspond à l'ensemble des états du système.

- A = A_1 x ... x A_n définit l'ensemble des actions jointes des agents, A_i est l'ensemble des actions locales de l'agent Ag_i.

- $P : S \times A \times S \longrightarrow [0, 1]$ est une fonction de transition.

- Ω_i est l'ensemble des observations possibles pour l'agent i

- $O : S \times A \times \Omega \rightarrow [0, 1]$ définit les probabilités d'observation (sur les actions et observations jointes)

- $R : S \times A \times S \rightarrow \mathbb{R}$ désigne la fonction de récompense.

Elle est définie sur l'espace S et fournit la récompense immédiate aux agents en fonction de l'action jointe émise dans le système.

Résoudre de manière optimale un DEC-MDP consiste à trouver une politique optimale jointe π= $<\pi_1, \pi_2$, ..., $\pi_n>$ composée d'un ensemble de politiques locales π_i. Une politique optimale jointe est une politique qui maximise l'utilité espérée du système. Elle fournit aux agents un comportement optimal, c'est-à-dire qu'à chaque étape de décision, chaque agent choisit l'action susceptible d'augmenter le plus le gain du système.

En raison du manque d'observabilité de l'état du système et des actions des autres agents, la résolution optimale de problèmes de contrôle décentralisé est beaucoup plus difficile que dans le cadre du contrôle centralisé. En effet, les performances du système dépendent de son état global et de l'action jointe exécutée. Cependant, lorsqu'un agent décide quelle action réaliser, il n'a qu'une connaissance partielle de ces informations.

Nous aborderons plus particulièrement l'utilisation des processus décisionnels de Markov pour la résolution des problèmes de décision multi-agents (DEC-MDP) de grande taille.

4 Modélisation du problème CTM par un DEC-MDP

Dans le problème "Contrôle d'un troupeau de moutons" l'environnement du système est une arène circulaire discrète. Chaque entité (berger ou mouton) est représenté par un couple d'entiers(x, y) dans un repère orthonormé d'origine le centre de l'arène circulaire.

Le modèle DEC-MDP pour le problème "Contrôle d'un troupeau de moutons" est décrit par le nombre d'agents, les états, les actions et les récompenses que nous détaillons dans ce qui suit :

Environnement : Notre environnement est totalement observable car l'agent perçoit toutes les informations nécessaires à la décision et déterministe car le prochain état de l'environnement est totalement déterminé par son état courant et par l'action exécutée.

Nombre d'agent : trois bergers.

Etat : Un état du MDP est une paire (X, Y) où :
- X représente le vecteur des abscisses des bergers et des moutons.
- Y représente le vecteur des ordonnés des bergers et des moutons.

Le MDP est constitué de tous les états possibles dans lesquels le système pourra se trouver.

Action : Les bergers et les moutons ne disposent que de quatre actions possibles correspondant à des déplacements d'une case vers le bas, vers la gauche, vers le haut et vers la droite.

Récompense : Dans le cas où on arrive à avoir tous les moutons dans l'enclos (but atteint), les agents (bergers) reçoivent une récompense positive.

5 Aspect combinatoire du CTM

L'état du MDP est constitué de l'ensemble des états de l'agent. Nous allons déterminer le nombre d'états pour estimer la faisabilité du calcul.

Notre environnement est une arène circulaire discrète. L'aire de cette arène (cercle) est le nombre de cases unités à l'intérieur de ce cercle. Chaque case dans le cercle à une aire d'une unité, on peut estimer le nombre total de cases par le calcule suivant :

$$Nb_cases \approx \pi * R^2 \tag{1}$$

Sachant que deux entites (berger ou mouton) ne peuvent pas se trouver à la même position, et que l'ordre dans lequel les moutons sont placés n'a pas d'importance, le nombre d'états du MDP peut être calculé par une combinaison comme décrite par la suite :

$$\begin{aligned} Nb_etats &= C_{Nb_cases}^{Nb_entites} * C_{Nb_entites}^{Nb_moutons} \\ &= \frac{Nb_cases!}{Nb_entites!(Nb_cases - Nb_entites)!} \\ &\quad * \frac{Nb_entites!}{Nb_moutons!(Nb_entites - Nb_moutons)!} \end{aligned} \tag{2}$$

Note que Nb_entite = Nombre du bergers + Nombre du moutons.

Nous donnons à titre indicatif le nombre d'états pour un environnement de rayon respectivement 6, 7, 8, 9 et 10

Rayon	6	7	8	9	10
Nombre de cases	113	154	201	254	314
Nombre d'états	2,2 10^{14}	8,45 10^{15}	1,03 10^{17}	8,8 10^{17}	6.11 10^{18}

Il est évident d'après ce tableau que notre problème CTM est un problème de grande taille.

6 Principe de résolution de CTM

Pour résoudre le problème du contrôle d'un troupeau de moutons et réduire l'espace d'états, nous avons définit les quatre techniques suivants :

- La décomposition concurrente qui consiste à partitionner l'espace géographique en trois sous espaces.
- La décomposition séquentielle qui consiste à ajouter un but intermédiaire : " une ligne de front ".
- L'agrégation en nuage dont le but est de remplacer la vue détaillée des bergers par une vue globale.
- L'agrégation de champs de vision : qui consiste à cnsidèrer comme un seul état les états où les moutons admettent un même comportement.

6.1 Technique de décomposition

Les méthodes de type "décomposition" forment une catégorie de méthodes de résolution approchée. Elles consistent à diviser le MDP global en sous MDP que l'on résout classiquement avant d'utiliser différentes techniques pour trouver une solution globale à partir des solutions locales. Suivant le type du MDP global, on distingue les méthodes de décomposition séquentielle et de décomposition concurrente (Cucala *et al.*, 2002).

6.1.1 Technique de décomposition concurrente dans CTM

Le principe de la décomposition concurrente est de définir les espaces géographiques de chaque berger. Dans le problème particulier du contrôle du troupeau des moutons il faut disposer d'un minimum de trois bergers pour encercler les moutons afin d'arriver au but (Flacher, 2005).

En regardant l'espace d'état comme étant un produit cartésien, on a décomposé l'espace géographique total (arène circulaire dont la surface $S_{arène} = \pi R^2$) en trois parties à peu près égales. Ainsi, la surface d'une zone de chaque berger $S_{ZoneAgent}$ est donnée par l'équation suivante :

$$S_{ZoneAgent} = \frac{S_{arène}}{3} = \frac{\pi R^2}{3} \tag{3}$$

Etant donné que chaque berger doit bénéficier d'un accès au but et qu'il doit avoir des espaces de recherche à peu près égaux, une décomposition triangulaire de l'espace géographique s'impose (Figure 2). Pour construire la zone noire du milieu, on doit calculer sa hauteur et sa base.

La surface du triangle est donnée par l'équation suivante :

$$S_{triangle} = \frac{base * hauteur}{2} \tag{4}$$

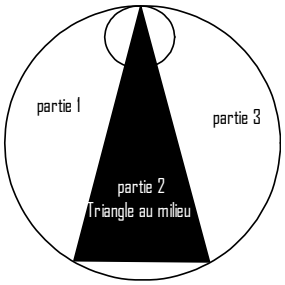


FIG. 2 – Décomposition triangulaire

Ainsi, les équations 3 et 4 donnent l'équation :

$$\frac{\pi R^2}{3} = \frac{base * hauteur}{2} \Rightarrow base = \frac{2\pi R^2}{3 * hauteur} \tag{5}$$

D'après la figure 2, nous pouvons estimer la hauteur par l'équation :

$$hauteur \simeq 2R \tag{6}$$

D'où, on peut conclure la base du triangle par l'équation suivante :

$$Base \simeq \frac{R\pi}{3} \tag{7}$$

Après le calcul de la base du triangle au milieu, on arrive donc a décomposer l'espace d'états du MDP initial en trois sous espaces d'états Zone_1, Zone_2 et Zone_3 (Figure 3). Chaque sous-espace d'états Zone_i est contrôlé par un seul berger. Les moutons peuvent se déplacer d'une zone à une autre.

Notre DEC-MDP pour les trois agents est alors un ensemble de trois MDP locaux. Chaque MDP local d'un agent A_i (berger_i) est défini par le tuple (S_i, A_i, P_i, R_i). On peut ainsi définir trois sous-MDP (MDP_1, MDP_2 et MDP_3).

On peut généraliser cette décomposition à n bergers. Ainsi, le DEC-MDP initial sera décomposé en n MDPs locaux. Chaque MDP local d'un agent A_i est défini par le tuple (S_i, A_i, R_i) telle que :

- S_i : l'ensemble des états du Berger_i.
- A_i : ensemble des actions possibles pour le Berger_i dans la Zone_i.
- R_i : S_i * A$i \rightarrow \mathbb{R}$ désigne la fonction de récompense immédiate au Berger_i.

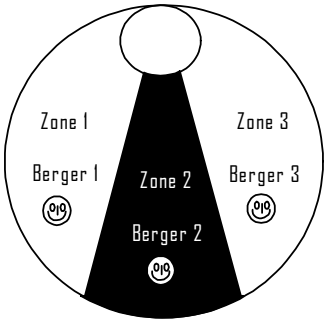


FIG. 3 – Une décomposition triangulaire de l'environnement

Maintenant que nous avons effectué la décomposition concurrente calculons le nombre d'états pour un MDP_i local. L'équation de calcul du nombre d'états est la même que celle de l'équation 2, sauf que pour une zone donnée le nombre de moutons varie de 0 à 6. Par conséquent, le $Nb_entites$ (moutons + 1 berger) varie de 1 à 7. D'où le nombre d'états possible pour une zone est donnée par l'équation 8.

$$Nb_etats = \sum_{Nb_entites=1}^{7} C_{Nb_cases}^{Nb_entites} * C_{Nb_entites}^{Nb_entites-1} \tag{8}$$

Puisque la surface d'une zone d'un agent est égale à $\frac{\pi R^2}{3}$(équation 3) le nombre de cases par zone est alors égal à $Nb_cases \simeq \frac{\pi R^2}{3}$

Ainsi, le nombre d'états, pour un MDPi, pour des environnement de rayons respectivement 6, 7, 8, 9 et 10 est donné par le tableau suivant :

Rayon	6	7	8	9	10
Nombre de cases	38	51	67	85	105
Nombre des états	1,5 10^7	1,3 10^8	9,7 10^8	5,4 10^9	2,4 10^{10}

On constate une nette diminution du nombre de cases ce qui implique une réducton d'un rapport d'environ 10^7 en nombre des états.

6.1.2 Technique de décomposition séquentielle dans CTM

Dans le cadre du CTM mais avec un seul berger, Vaughan décompose le problème en deux sous problèmes indépendants (Vaughan *et al.*, 2000). Le premier sous-problème consiste à s'approcher suffisamment du troupeau pour le pousser sans le disperser. Le second sous-problème permet de pousser le troupeau en direction du but. Il propose pour cela que son agent soit attiré vers le centre du troupeau.

Flacher gère le problème de Vaughan dans le cas du groupe d'agents par l'utilisation de plusieurs bergers(Flacher, 2005). Il a mentionné qu'il faut garantir que les bergers s'éloignent les uns des autres et qu'ils

se séparent de façon à former une ligne de front, afin de contenir le troupeau au lieu de le disperser. Pour séparer les bergers de façon à ce qu'ils forment une ligne de front, il propose de les repousser de manière perpendiculaire à leur distance au but tout en préservant la ligne de front.

Dans le cadre de notre travail, nous proposons de décomposer le MDP local initial en deux MDP indépendants :

- MDP "ligne de front" qui a comme but d'arriver à l'état où les bergers forment une ligne de front (figure 4).
- MDP "but final" dont l'objectif est d'arriver au but final tout en respectant la contrainte ligne de front.

La ligne de front d'un berger est définie par la droite passant par le berger et perpendiculaire à la droite reliant le berger et l'enclos.

Nous formalisons ainsi le MDP "ligne de front" pour un berger i comme suit :

- S_i : l'ensemble d'états tels qu'il existe au moins un mouton en dessous de la ligne de front.
- A_i : ensemble des actions possibles pour le berger i dans sa zone.
- $R_i : S_i \times A_i \rightarrow \mathbb{R}$ désigne la fonction de récompense immédiate du berger i.

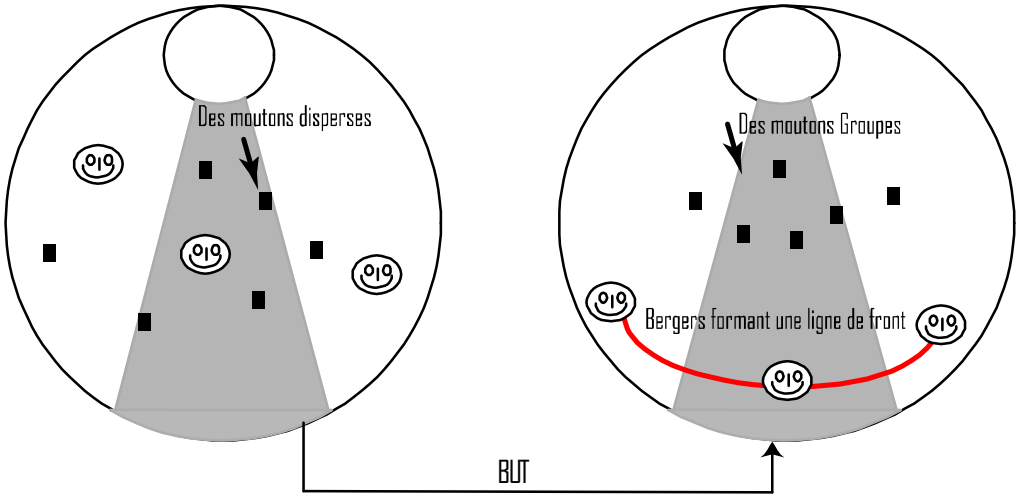


FIG. 4 – Une décomposition séquentielle : MDP en ligne de front

Le MDP "but final" pour un berger i sera formalisé par :

- S_i : l'ensemble d'états tels que tous les moutons sont au dessus de la ligne de front.
- A_i : ensemble des actions possibles pour le berger dans sa zone.
- $R_i : S_i \times A_i \rightarrow \mathbb{R}$ désigne la fonction de récompense immédiate du berger i.

6.2 Technique d'agrégation pour CTM

Le principe de cette technique est de regrouper les états de base ayant des caractéristiques communes au sein d'un état global (Laroche, 2000). Ainsi, au lieu de résoudre un MDP de base, on résout un MDP global. L'intérêt est de réduire le plus possible l'espace d'états, afin d'accélérer la résolution du MDP (Cucala *et al.*, 2002).

La principale difficulté consiste à détecter les caractéristiques utilisées pour regrouper les états et l'influence de ces caractéristiques sur la qualité de la politique obtenue. On peut déduire que chaque application admet ces propres méthodes d'agrégation d'états.

Dans cette partie, nous proposons deux méthodes d'agrégation d'état pour réduire le nombre d'états du MDP "but final", à savoir l'agrégation en nuage et l'agrégation de champs de vision que nous détaillons dans ce que suit :

6.2.1 Agrégation en nuage.

Afin de minimiser le nombre d'états exprimant tous les cas de figure de dispersion des moutons dans la zone de chaque agent et ainsi améliorer la rapidité d'apprentissage, nous proposons l'idée suivante : l'agent n'a pas une description détaillée de chaque mouton mais une vue globale de l'ensemble du troupeau de moutons. Dans cette optique, l'utilisation du barycentre est envisagée pour le regroupement des moutons et l'écart type pour leur répartition dans une zone.

En effet, la notion d'écart type, sur l'axe des abscisses et l'axe des ordonnés, permet de caractériser la répartition géographique des moutons autour du barycentre. Ce qui permet de mesurer la répartition des moutons. Plus l'écart type est faible, plus les moutons sont proches et inversement lorsque l'écart type est grand les moutons sont éloignés. Nous considérons que les moutons forment un groupe s'ils vérifient la condition d'écart type suivant :

$$\begin{aligned} x_{EcartType} &< seuil_EcartType \\ y_{EcartType} &< seuil_EcartType \end{aligned}$$

Avec ($x_{EcartType}$, $y_{EcartType}$) étant l'écart type des moutons et $seuil_EcartType$ est un paramètre définit expérimentalement.

Deux états $e1$ et $e2$ ayant une même position du berger et des barycentres des moutons respectifs $B1$ et $B2$ sont identiques s'ils vérifient la condition de nuage suivant :

e1 et e2 vérifient la condition d'écart type et Distance(B1,B2) $<$ seuil_Barycentre

Nous indiquons que $seuil_Barycentre$ et $seuil_EcartType$ ne sont pas des constantes mais plutôt des variables proportionnelles à la distance entre le berger et le barycentre des moutons. Ainsi,

$$seuil_Barycentre = K_{barycentre} * distance(berger, barycentre) \tag{9}$$

$$seuil_EcartType = K_{ecarttype} * distance(berger, barycentre) \tag{10}$$

où $K_{barycentre}$ et de $K_{ecarttype}$ sont des constantes fixées expérimentalement.

6.2.2 Agrégation : Champs de vision

Notre objectif est toujours la minimisation du nombre d'états. Vue qu'on estime que la réaction des regroupements des moutons appartenant au même champ de vision d'un berger est la même, on peut considérer leur état comme un seul état.

Un champ de vision est une portion de l'espace dont tous les points peuvent être perçus par un capteur. Dans notre cas, le champ de vision d'un berger est un cône de sommet principal le berger et d'angle au sommet $alpha$. La zone de chaque berger est décomposée statiquement en champs de vision d'angles égaux. Les champs de vision sont disposés à droite et à gauche du champ de vision milieu. Le champ de vision milieu et celui dont la bissectrice est la droite reliant le berger et le centre de l'enclos. L'angle des champs de vision est déterminé expérimentalement.

Si les moutons appartiennent à un même champ de vision, l'état est représenté par les coordonnées du berger et l'angle $alpha$.

Par ailleurs, si l'agent est en dehors du champ de visibilité des moutons, il peut se déplacer sans modifier leurs positions. Ainsi on peut diviser l'environnement pour un agent en deux zones : une zone de répulsion et une zone libre (figure 5) :

- La zone de répulsion est celle où le déplacement de l'agent a un effet sur les mouvements des moutons.
- La zone libre est celle où l'agent peut se déplacer sans influencer le mouvement des moutons.

Ainsi, deux états e1 et e2 sont identiques s'ils vérifient la condition de champ de vision à savoir :

$e1$ et $e2$ appartiennet à un même champ de vision du berger et à une même zone.

Pour conclure, pour un agent i, les états du MDP après agrégation est donné par l'organigramme (figure 6) :

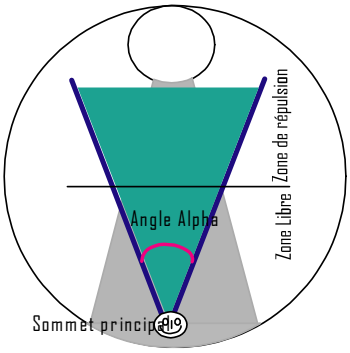


FIG. 5 – Champs de vision et zone libre ou zone de répulsion

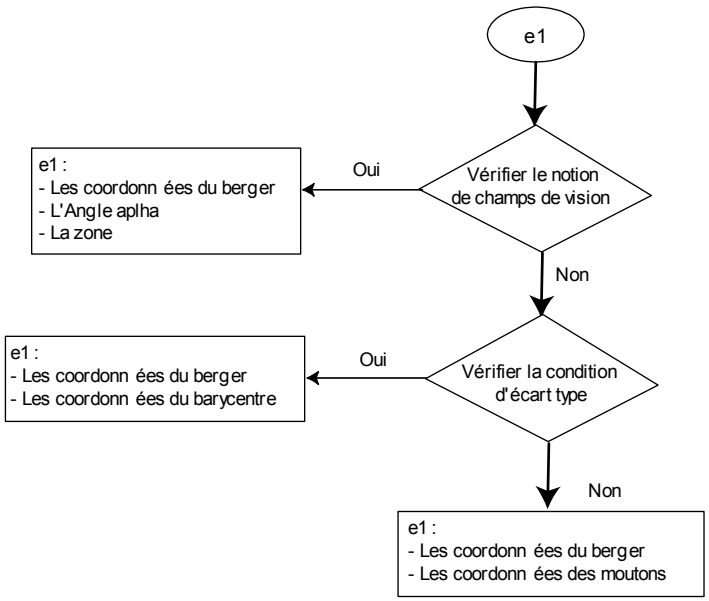


FIG. 6 – Les états d'un agent

7 Etude expérimentale

7.1 Effet de la décomposition concurrente

On remarque qu'avec cette méthode on arrive à atteindre un résultat. Ce qui n'était pas possible avec l'approche classique. Néanmoins, on n'a pas réussit à atteindre la stabilité d'apprentissage malgré l'importance de la durée d'apprentissage qui a dépassé 1 mois

7.2 Effet de la décomposition séquentielle

On distingue un pourcentage de réussite de 97% après une durée d'apprentissage de 21 jours. Cette décomposition donne une amélioration du résultat. Avec cette décomposition on a réussi à atteindre une stabilité après 20 jours.

7.3 Effet d'agrégation en nuage

Durant l'expérimentation nous avons fixé la valeur de $K_{barycentre}$ à 0.2 et nous avons essayé de déterminer la meilleure valeur $K_{ecarttype}$.

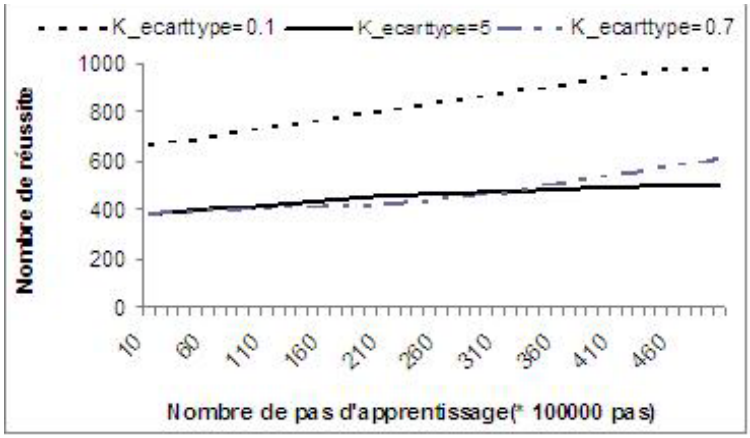


FIG. 7 – Agrégation : Nuage

Rappelons que EcartType = $K_{ecarttype}$ *distance(Barycentre, Berger). D'après la courbe dans la figure 7 on constate que :

- Pour $K_{ecarttype}$=0,1 nous obtenons un excellent taux de réussite en dépit d'une durée de 20 jours.
- Pour $K_{ecarttype}$= 5 nous obtenons les mêmes résultats que ceux obtenus avec la technique du barycentre.
- Pour $K_{ecarttype}$=0.7 le taux de réussite s'est amélioré pour atteindre 61% pour une durée d'apprentissage de 14 jours.

Avec cette technique un problème persiste est celui de déterminer le meilleur $K_{ecarttype}$.

7.4 Effet d'agrégation de champs de vision

Nous allons tester la technique introduite auparavant qui est le champ de vision. Suite à un ensemble de tests réalisés, avec différentes valeurs pour l'angle $alpha$, la valeur 30 a paru la plus prometteuse.

D'après la courbe de la figure 8 et avec un angle $alpha$ de 30', nous avons obtenu un taux de réussite proche de 72% pour une durée d'apprentissage ne dépassant pas 2 heures ce qui représente une nette amélioration par rapport aux autres techniques.

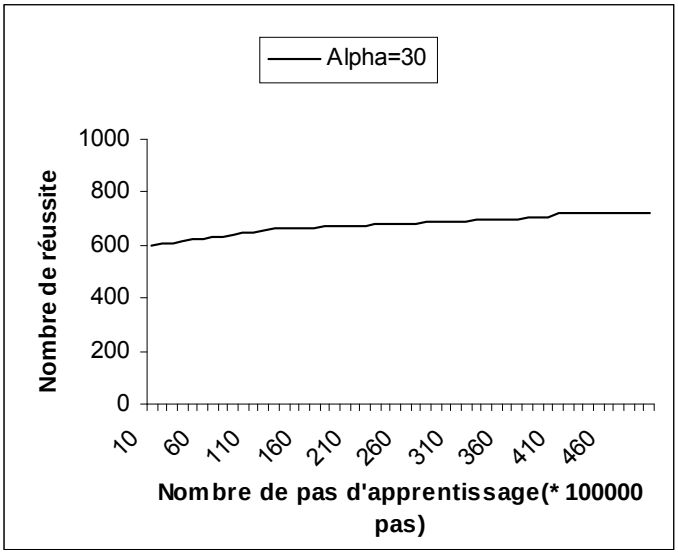


FIG. 8 – Agrégation : Champs de vision

8 Conclusion

Les problèmes de grande taille peuvent être résolus par des méthodes de décompositions de problèmes et d'agrégation d'états. Ces méthodes sont relatives au domaine d'application. Dans le cadre de notre application, contrôle d'un troupeau de mouton, nous avons effectué deux types de décomposition à savoir la décomposition concurrente et la décomposition séquentielle. Pour la décomposition concurrente nous avons partitionné l'espace géographique en trois zones à peu près égales. Concernant la décomposition séquentielle, nous avons transformé notre MDP en deux MDPs indépendants en ajoutant un but intermédiaire "ligne de front". Ce dernier représente l'état final du premier MDP et l'état initial du second MDP.

L'étude expérimentale a montré l'intérêt de l'approche par décomposition. En effet, la technique de décomposition concurrente a permis d'obtenir une solution à notre problème, qui été préalablement sans solution, avec un taux de réussite égal à 69% mais dans un délai d'apprentissage assez grand 30 jours. La technique de décomposition séquentielle a conduit à une amélioration au niveau du taux de réussite (97%). Mais, le délai d'apprentissage reste toujours élevé soit 21 jours.

Nous avons par ailleurs utilisé l'approche d'agrégation qui permet de modifier le champ visuel d'un agent (berger) d'un champ visuel détaillé en un champ visuel global en utilisant les notions du nuage et champs de vision. Ces techniques nous ont permis de réduire le temps d'exécution en passant de 21 jours à deux heures. Par contre, nous avons noté une baisse dans le taux de réussite (de 97% à 72%).

Références

BERNSTEIN D., S Z. & IMMERMAN N. (2002). The complexity of decentralized control of mdps. In R. TEULIER, Ed., *In Mathematics of Operations Research*.

CUCALA L., GARCIA F. & SABBADIN R. (2002). Résolution approchée de processus décisionnels markoviens de grande taille. In *Actes du 13ème congrès Reconnaissance des Formes et Intelligence Artificielle (RFIA 2002), Angers*.

FLACHER F. (2005). Générationascendantede coordination spatiale versuneconception automatisée du contrôlede coordination spatiale. In *Actes d'IC'97*.

LAROCHE P. (2000). Processus décisionnels de markov appliqués à la planification sous incertitude. In *Thèse de doctorat, Université Henri Poincaré Nancy 1*.

VAUGHAN T. R., SUMPTER N., HENDERSON J. V., FROST A. & CAMERON S. (2000). Experiments in automatic flock control. In *Robotics and Autonomous Systems*.

Planification à base d'ordres topologiques pour la résolution des POMDPs

Jilles S. Dibangoye[1,2], Brahim Chaib-draa[1], Abdel-Illah Mouaddib[2]

[1] Laboratoire Damas
Université Laval - Québec Q.C., Canada
jilles-steeve.dibangoye.1@ulaval.ca

[2] Laboratoire Greyc – UMR 6072,
Université de Caen, France

Résumé : Bien que les processus décisionnels de Markov partiellement observables (POMDPs) aient reçu beaucoup d'attention au cours des dernières années, à ce jour, la résolution des problèmes réels reste un sérieux défi. Dans ce contexte, les techniques d'accélération des méthodes fondamentales ont été un des principaux axes de recherche. Parmi elles, les algorithmes ordonnés suggèrent des solutions au problème de l'ordre des *mises à jour* de la fonction de valeur. Ces techniques permettent notamment de réduire considérablement le nombre de mises à jour requises, mais en contrepartie, elles impliquent des coûts supplémentaires. Dans ce papier, nous présentons une nouvelle approche ordonnée : *la planification à base d'ordres topologiques* (TOP). Cette approche exploite les relations de causalité entre états afin de faire face à deux problématiques : (1) la détection de la structure d'un POMDP comme moyen de surmonter à la fois la malédiction de la dimension et celle de l'historique ; (2) la réduction des mises à jour inutiles et le déterminisme d'une fonction de valeur approximative sur la base d'ordres topologiques induits par la structure sous-jacente du POMDP. Les expérimentations prouvent que TOP est compétitif comparativement aux meilleurs algorithmes actuels. **Mots-clés** : POMDPs, Prise de décision séquentielle, Ordre topologique.

1 Introduction

Les POMDPs offrent un cadre robuste pour la planification dans les domaines impliquant états cachés et incertitude sur les effets des actions. Toutefois, malgré un effort soutenu dans le développement d'algorithmes de plus en plus efficaces, *e.g.,* (Roy *et al.*, 2005; Poupart & Boutilier, 2004; Pineau *et al.*, 2003; Cassandra *et al.*, 1997), les aspects de complexité et de calculabilité limitent l'utilisation des POMDPs dans les problèmes réels. Les limitations de ces techniques sont, dans une large mesure, une conséquence de deux problèmes étroitement liés : *la malédiction de la dimension* (Bellman, 1957), où la dimension du problème de planification est déterminée par le nombre d'états ; et *la malédiction de l'historique* (Pineau *et al.*, 2003), où le nombre d'historiques croît de façon exponentielle avec l'horizon de planification. Bon nombre de stratégies ont été proposées afin de réduire le nombre d'historiques considérés pour l'obtention d'une fonction de valeur exacte (Cassandra *et al.*, 1997) ou approximative (Pineau *et al.*, 2003). Les approches exactes de déterminisme d'une fonction de valeur ε-optimale sur l'espace entier des croyances, espace continu, utilisent des techniques d'élagage afin de réduire l'ensemble des historiques. Néanmoins, en raison de la continuité de l'espace des croyances, ces approches se sont révélées inefficaces en pratique. Les méthodes parmi les plus prometteuses dans le déterminisme d'une fonction de valeur approximative, pour des domaines de très grande dimensions sont basées sur l'opérateur de *mise à jour à base de points*. En particulier, Pineau *et al.* (2003) ont abandonné la mise à jour de la fonction de valeur sur l'ensemble des points de croyance en faveur de celle d'un seul point à la fois. L'algorithme résultant nommé *itération de valeurs à base de points* (PBVI) utilise alors un opérateur de mise à jour polynomial en lieu et place d'un opérateur exponentiel. Toutefois, la majorité des méthodes à base de points, y compris PBVI, restent limitées par la malédiction de la dimension. Comme nous l'avons déjà mentionné, certains algorithmes sont incapables de passer à l'échelle car ils calculent leur fonction de valeur exacte ou approximative sur l'ensemble entier des états de croyance. Cependant, dans les applications réelles, le déterminisme d'une fonction de valeur sur l'espace entier des états de croyance est bien souvent inutile. Selon Roy *et al.* (2005), l'espace des états de croyance accessible par simulation est souvent associé à un sous-espace structuré, de petite dimension, inclut dans un espace de plus grande dimension. Par conséquent, la recherche et l'exploitation de la structure

des POMDPs peuvent améliorer les algorithmes fondamentaux. Malheureusement, alors que des approches structurées, telles que celles qui sont proposées par Roy *et al.* (2005); Poupart & Boutilier (2004), peuvent conduire à une réduction exponentielle de la dimension de certains POMDPs, leurs mécanismes algorithmiques sont très délicats. Bien qu'utiles et intéressantes, ces approches n'ont aucun lien direct avec la nôtre, pour cette raison nous ne poursuivrons pas la comparaison.

La raison d'être de notre approche est de faire face aussi bien la malédiction de la dimension quà la malédiction de l'historique, et cela dans un même mécanisme algorithmique. TOP généralise la relation de causalité entre les états, relation qui a été utilisée avec succès en MDPs (Dai & Goldsmith, 2007; Bonet & Geffner, 2003; Abbad & Boustique, 2003). Il atténue la malédiction de la dimension en cloisonnant les états en groupe d'états mutuellement dépendants, *i.e,* couches. Le graphe de transition sur ces couches constitue la structure du problème. Puis, il n'effectue de planification que sur l'ensemble des couches dites *pertinentes*. Il surmonte la malédiction de l'historique en se limitant à un petit nombre de couches à chaque *simulation* (de trajectoires). TOP traverse simultanément l'espace des états de croyance et l'espace des états du MDP sous-jacent, et cela suivant des ordres topologiques. Il n'effectue de mises à jour que si celles-ci sont susceptibles de changer la fonction de valeur de façon non-triviale. Cela permet d'éviter les problèmes de mises à jour inutiles. Enfin, TOP est capable d'identifier les couches où la fonction de valeur a convergé et donc de re-diriger la recherche vers des couches non *ou* pas suffisamment explorées.

Le reste de ce papier est organisé comme suit : dans la Section 2, nous couvrons tout d'abord le contexte général des POMDPs et passons en revue les travaux parmi les plus récents y compris les techniques ordonnées ; Ensuite, nous présentons notre nouvelle approche ordonnée en Section 3, et discutons de sa mise en œuvre en Section 4. Enfin, nous présentons une évaluation empirique de TOP, et démontrons ses forces et faiblesses par rapport à d'autres algorithmes bien connus, et cela sur un grand nombre de bancs d'essai.

2 Contexte général et travaux connexes

Nous commençons avec une vue d'ensemble des MDPs, POMDPs puis des espaces d'états de croyance. Nous offrons ensuite une courte introduction à l'état de l'art des algorithmes de résolution des POMDPs.

Un processus décisionnel de Markov (MDP) est un n-uplet (S, A, T, R, γ) où : S est un ensemble discret et fini d'états ; A est un ensemble discret et fini d'actions ; $T(s'|s, a)$ est une fonction de transition, *i.e.,* la probabilité de transition d'un état s à un état s' après l'exécution de l'action a ; $R(s, a)$ est une valeur réelle décrivant la récompense reçue après avoir exécuté l'action a dans l'état s ; γ est le facteur de décompte. Le cadre des MDPs a ensuite été généralisé aux processus décisionnels de Markov partiellement observables (POMDPs) pour la planification dans des domaines impliquant états cachés et incertitude sur les effets des actions. Plus formellement, un POMDP est un n-uplet $(S, A, T, R, \Omega, O, \gamma, b_0)$, où S, A, T, R et γ sont identiques au cadre des MDPs ; Ω est une ensemble discret et fini d'observations ; $O(o|s', a)$ est la probabilité d'observer o après exécution de l'action a et l'accès à l'état s' ; Un des aspects importants en POMDPs est la notion d'états de croyance noté b décrivant une distribution de probabilité sur l'ensemble des état du système, et constituant la statistique suffisante pour un historique donné ; b_0 désigne ainsi l'état de croyance initial, *i.e.,* la croyance de l'agent quant à son état initial dans le système. L'état de croyance suivant b, noté $b' = \tau(b, a, o)$, est mise à jour comme suit : $b'(s') = \frac{O(o|s',a)\sum_s b(s)T(s'|s,a)}{pr(o|b,a)}$, avec $pr(o|b, a) = \sum_s b(s) \sum_{s'} T(s'|s, a)O(o|s', a)$. étant donné un POMDP, l'objectif est de trouver une fonction de valeur V qui associe à tout état de croyance un hyperplan. En outre, comme la fonction de valeur V a été prouvée linéaire par morceaux et convexe (Sondik, 1978), elle peut être représentée comme un ensemble d'hyperplans $V = \{\alpha_1, \alpha_2, \cdots, \alpha_m\}$ dit α-vecteurs, de sorte que $V(b) = \max_i (\alpha_i \cdot b)$. Enfin, la fonction de valeur V peut-être iterativement calculée en utilisant l'opérateur de mise à jour à base de points (Pineau *et al.*, 2003; Spaan & Vlassis, 2005; Smith & Simmons, 2004, 2005) :

$$\begin{array}{lcl} g^{\alpha}_{a,o}(s) & = & \sum_{s'} O(o|s', a)T(s'|s, a)\alpha_i(s') \\ g^{b}_{a} & = & r_a + \gamma \sum_o \arg\max_{g^{\alpha}_{a,o}:\ \alpha \in V} (b \cdot g^{\alpha}_{a,o}) \\ backup(b) & = & \arg\max_{g^{b}_{a}:\ a \in A} (b \cdot g^{b}_{a}) \end{array} \quad (1)$$

où $r_a(s) = R(s, a)$ est un vecteur de réprésentation de la fonction de récompense.

2.1 Algorithmes approximatifs

L'obstacle dû aux calculs résultant du nombre de vecteurs, qui sont générés pour le déterminisme d'une fonction de valeur optimale sur l'espace entier des états de croyance, a conduit au développement d'une vaste variété de méthodes approximatives. Au lieu de calculer une fonction de valeur optimale sur l'ensemble de l'espace des états de croyance, une approximation possible est de le faire uniquement sur un nombre fini d'états de croyance (Lovejoy, 1991). En conséquence l'opérateur de mise à jour est polynomial et la taille de la fonction de valeur est bornée par le nombre d'états de croyance. Néanmoins, une fonction de valeur optimale sur un ensemble discret d'états de croyance B est difficilement généralisable aux états de croyance hors de B. En revanche, les algorithmes de mise à jour à base de points mettent à jour uniquement les états de croyance qui sont *accessibles* en partant de b_0 (Pineau *et al.*, 2003; Spaan & Vlassis, 2005; Smith & Simmons, 2004, 2005). En mettant à jour ces états de croyance accessibles, il est empiriquement et théoriquement prouvé que la fonction de valeur se généralisera aux états de croyance hors de B.

En particulier, PBVI calcule alternativement les deux, un ensemble discret B d'états de croyance accessibles et la fonction de valeur optimale correspondante (Pineau *et al.*, 2003). L'algorithme débute par la construction de l'ensemble initial $B_0 = \{b_0\}$ des états de croyance, puis étend B_0 à B_1 avec l'ensemble des états de croyance les plus distants parmi les successeurs immédiats des états de croyance de B_0 et calcule une fois de plus la fonction de valeur optimale sur B_1. Ce processus se poursuit jusqu'à ce qu'un critère d'arrêt ait atteint. Un certain nombre d'inconvénients limitent cependant les capacités de l'algorithme PBVI à passer à l'échelle. Tout d'abord, l'étape d'expansion des états de croyance engage un grand nombre de ressources. En effet, cette étape consiste à comparer $|B||A||O|$ successeurs immédiats selon leur distance, *i.e.,* métrique L_2. Par conséquent, dans le pire des cas, cette sous-routine requiert $O(|B|^2|A||O||S|)$ opérations. Pour faire face à ce point faible, Pineau *et al.* (2003) ont proposé l'échantillonnage d'un successeur pour chaque paire, état de croyance – action, cela conduit à la réduction de la complexité de la phase d'expansion à $O(|B|^2|A||S|)$. Le second point de faiblesse réside dans la phase de mise à jour. Cela est principalement dû au fait que PBVI met à jour tous les états de croyance de B à chaque itération, même si cela est inutile ou redondant pour certains d'entre eux.

Perseus proposé par Spaan & Vlassis (2005), est un autre algorithme à base de points. Il est dans l'esprit assez proche de PBVI. À la différence que l'étape d'expansion de PBVI est remplacée par une marche aléatoire à partir de l'état de croyance initial b_0, et au lieu de mettre à jour tous les états de croyance B à chaque itération Perseus le fait seulement pour un sous-ensemble de B. Ce sous-ensemble correspond aux états de croyance dont les valeurs n'ont pas été améliorées à la dernière mise à jour. Comme Perseus met souvent à jour un sous-ensemble d'états de croyance, il peut soit converger plus rapidement ou utiliser un plus grand nombre d'états de croyance afin d'améliorer la qualité de sa fonction de valeur. Le principal problème de Perseus réside dans la lenteur de sa convergence dans certains domaines. Bien que la marche aléatoire dirige la recherche vers des espaces les plus probablement accessibles, il n'existe aucune garantie que ces domaines offriront les récompenses utiles pour l'amélioration de la fonction de valeur. En outre, en traversant l'espace des états de croyance suivant une marche aléatoire sont comportement est aléatoire tout comme ses performances dans certains des domaines que nous avons testés.

De cette famille d'algorithmes, *heuristic search value iteration* (HSVI) a montré les meilleures performances et cela sur les bancs d'essai parmi les plus grands (Smith & Simmons, 2004, 2005). HSVI procède comme suit : il crée une séquence d'états de croyance, *i.e.,* trajectoire d'états de croyance, en se fondant à la fois sur la borne supérieure $\bar{V}$ et la borne inférieure $\underline{V}$ de la fonction de valeur V ; En partant avec l'état de croyance initial b_0, le prochain état de croyance qu'il sélection est le successeur du précédant maximisant l'écart entre la borne supérieure et la borne inférieure. L'expansion des états de croyance prend fin quand un critère d'arrêt est atteint, s'ensuit la mise à jour de la fonction de valeur dans le sens inverse de la génération des états de croyance. Malheureusement, aussi bien le maintien de la borne supérieure et la sélection de l'état de croyance suivant engendrent des coûts additionnels non négligeables. Par conséquent, le gain résultant de la réduction du nombre de mises à jour ne se manifeste pas pleinement dans le temps d'exécution total. Tout au long de l'exécution de l'algorithme, l'écart entre la borne inférieure et la borne supérieure à l'état de croyance initial b_0 décroît jusqu'à atteindre une valeur inférieure ou égale à un nombre réel ε. L'algorithme HSVI s'arrête alors. Néanmoins d'un point de vue pratique la clôture de cet écart est quasiment impossible à moins que l'écart initial soit relativement petit, comme nous le constatons dans les bancs d'essai de type Rocksample.

2.2 Algorithmes à base de simulation de trajectoires

Afin de surmonter le problème de la mise à jour systématique de tous les états de croyance à chaque itération, les chercheurs ont tourné leur attention vers des techniques à base de simulations de trajectoires. Il s'agit de méthodes où l'agent simule des interactions avec l'environnement et crée des trajectoires d'états de croyance. Parmi d'autres méthodes de cette classe, l'algorithme *forward search value iteration* (FSVI), récemment proposé par Shani *et al.* (2007), a démontré de très bonnes performances. FSVI utilise la Q-valeur du MDP sous-jacent comme heuristique pour l'exploration de l'espace des états de croyance. Dans ce dessein, il maintient à la fois un état s du MDP sous-jacent et un état de croyance b, ce qui permet la sélection d'une action a fondé sur l'état sous-jacent s et la Q-valeur du MDP sous-jacent. FSVI procède par la suite à la sélection de l'état suivant s' et l'observation o, en les échantillonnant de $T(\cdot|s,a)$ et $O(\cdot|a,s')$, respectivement. L'avantage de FSVI réside dans le faible coût résultant du calcul de la Q-valeur du MDP sous-jacent et donc la sélection des actions. Toutefois, un des inconvénients majeurs de FSVI repose sur le fait qu'il n'est pas en mesure de reconnaître les trajectoires qui permettront d'améliorer l'information sur l'état dans lequel il se trouve. En effet, en faisant usage de la Q-valeur optimale du MDP sous-jacent, FSVI limite sa capacité à créer des trajectoires qui visiteront des états susceptibles de fournir des observations utiles. Encore plus important, il est incapable de réorienter la recherche vers les états qui sont probablement inaccessibles suivant la Q-valeur du MDP sous-jacent, et cela même si ces derniers peuvent fournir des récompenses utiles pour la qualité de la fonction de valeur. En conséquence, FSVI peut conduire à une fonction de valeur sous-optimale dans certains domaines. Néanmoins, dans les domaines qui ne nécessitent pas la collecte d'information sur le long terme et/ou une exploration plus large de l'espace de croyance, FSVI est très compétitif. Enfin, même une simple politique d'exploration très conservative, *e.g.,* ε-gourmand, peut compenser le manque d'exploration.

2.3 Algorithmes ordonnés

Les techniques ordonnées visent à résoudre le problème du déterminisme d'une fonction de valeur approximative à la fois rapidement et avec précision. L'idée d'organiser les mises à jour dans un bon ordre a déjà été examinée auparavant par Shani *et al.* (2006); Virin *et al.* (2007) et dans une certaine mesure par Shani *et al.* (2007); Smith & Simmons (2004, 2005).

Shani *et al.* (2006) ont introduit l'algorithme *prioritized value iteration* (PVI). Cet algorithme utilise un ensemble fixe d'états de croyance et met à jour ces états de croyance suivant l'erreur de Bellman, défini comme suit : $e(b) = \max_a [r_a \cdot b + \sum_o pr(o|b,a)V(\tau(b,a,o))] - V(b)$. PVI met à jour à chaque étape l'état de croyance $b \in B$ qui améliorera plus probablement la fonction de valeur. Afin d'y parvenir, Shani *et al.* (2006) ont suggéré, dans la version naïve de PVI, de choisir l'état de croyance dont l'erreur de Bellman est la plus grande. L'algorithme s'arrête lorsqu'il ne parvient pas à trouver un état de croyance dont l'erreur de Bellman soit supérieure à zéro. En mettant à jour les états de croyance suivant l'ordre décroissant de leur erreur de Bellman, il est probable que certains états de croyance soient plus souvent mise à jour que d'autres. Cela peut conduire à des gains significatifs en termes de performances. Toutefois, il est un point qui limite l'usage de PVI. En effet, le calcul de l'erreur de Bellman suite à chacune des mises à jour induit des efforts considérables. La complexité de la mise à jour de l'erreur de Bellman est non négligeable en comparaison à la complexité de l'opérateur de mise à jour à base de points : dans le pire des cas, ils requièrent $O(|B||A||O||S|)$ et $O(|S|^2|V||A||O|)$ respectivement. Pour surmonter ce problème, Shani *et al.* (2006) calculent l'erreur de Bellman uniquement pour un sous-ensemble $\tilde{B}$ des états de croyance échantillonnés, réduisant ainsi la complexité de mise à jour de l'erreur de Bellman à $O(|\tilde{B}||A||O||S|)$.

Dans le même ordre d'idées, Virin *et al.* (2007) ont proposé un algorithme ordonné, nommé *soft cluster based value iteration* (SCVI). Ce dernier est basé sur la méthode *generalized prioritized sweeping* introduite en MDPs par Wingate & Seppi (2005). Afin d'alléger la complexité relative au déterminisme d'une bonne séquence de mise à jour, de façon similaire à FSVI, ils utilisent un algorithme de *clustering* basé sur la fonction de valeur optimale du MDP sous-jacent. SCVI met ensuite à jour les états de croyance dans l'ordre décroissant d'association au *cluster* courant. Cette méthode a l'avantage d'éviter le tri systématique de la file d'attente des états de croyance requis dans le cas de PVI. Cependant, une fois de plus il est très probable qu'un état de croyance soit associé à plusieurs clusters, chaque état de croyance peut alors être mise à jour à chaque itération, et cela même si c'est inutile.

Enfin, on peut voir les algorithmes HSVI et FSVI comme des méthodes ordonnées. En effet, tous deux procèdent par simulation de trajectoires d'états de croyance suivant des heuristiques afin d'explorer l'espace des états de croyance et effectuent des mises à jour par chaînage arrière. La réalisation des mises à jour dans

l'ordre inverse des trajectoires a une importance cruciale. Comme nous le verrons plus tard, les fonctions de valeur et les états de croyance sont liés par une dépendance causale. À ce titre, la fonction de valeur d'un état de croyance successeur peut contribuer dans la fonction de valeur de l'état de croyance courant. Ainsi, la mise à jour prioritaire des états de croyance successeurs peut se traduire par un gain substantiel en performance.

3 Motivation

Tous les algorithmes ci-dessus font usage de l'opérateur de mise à jour à base de points sur un ensemble spécifique d'états de croyance afin de surmonter avec succès à la malédiction de l'historique. Jusqu'à présent cependant, toutes ces approches ont été victimes de la malédiction de la dimension lorsqu'ils sont confrontés aux domaines à très grands espaces d'états. Aucun des algorithmes énumérés ci-dessus n'utilise les caractéristiques intrinsèques des POMDPs. Ils ne considèrent guère les dépendances causales entre états notamment par le biais de la représentation graphique d'un POMDP, représentation qui selon Littman *et al.* (1995) est une propriété intrinsèque des MDPs donc des POMDPs, et potentiellement décide la complexité de leur résolution. Par exemple, la Figure 1 décrit un simple exemple de POMDP proposé par Pineau (2004), ainsi que la représentation graphique G associée. Examinons le problème à 5 états illustré en Figure 1. L'agent débute dans un état quelconque avec égale probabilité. Une fois dans ces états, la fonction d'observation (bruitée) fournit l'état actuel. En exécutant l'action a_1, l'agent se déplace aléatoirement entre les états s_1 et s_2, tandis qu'en exécutant l'action a_2 l'agent se déplace de vers les états s_3 et s_4. L'état s_5 est un état puits. La récompense est une fonction telle qu'il est bon ($+100$) de transiter vers l'état (s_3) et mauvais (-100) de transiter vers l'état s_4. La récompense est nulle partout ailleurs.

En observant ce POMDP, il nous apparaît que seul deux des composantes dans tout α-vecteur seront améliorées lors de l'application itérative des mises à jour, *i.e.,* les composantes associées aux états s_1 et s_2. Cela repose sur le fait que les composantes associées aux états s_3, s_4 et s_5 ont obtenu leurs valeurs optimales (respectivement $+100$, -100 et 0), à la première mise à jour. Par conséquent, certaines composantes des α-vecteurs peuvent avoir convergé tandis que d'autres n'y parviendront qu'après plusieurs mises à jour supplémentaires. Donc, ni les algorithmes exactes ni les méthodes approximatives cités plutôt n'échappent aux innombrables mises à jour parfois redondantes et/ou inutiles. De plus, ces algorithmes font l'usage abusif de toutes les composantes des α-vecteurs à chaque itération, bien qu'en pratiques certaines de ces composantes puissent avoir déjà atteint leurs valeurs optimales. Néanmoins, très peu d'intérêt a été accordé à ce point. À ce jour, toutes les approches tentant de résoudre ce POMDP s'y prendraient comme pour la résolution d'un quelconque POMDP à 5 états. Ils appliqueraient essentiellement la même stratégie que celle qu'ils appliqueraient pour un POMDP dont la représentation graphique correspond à une 5-clique. Ainsi, les stratégies de base de ces algorithmes n'ont aucune sous-routine intelligente, capable de distinguer différents POMDPs et d'utiliser différentes stratégies pour les résoudre. En s'inspirant de ces observations, nous voulons concevoir un algorithme général qui soit capable de découvrir la complexité intrinsèque des différents POMDPs par le biais de leur structure, afin d'utiliser différentes stratégies de mise à jour pour des POMDPs disposant de différentes propriétés structurelles.

4 Planification à base d'ordres topologiques

Dans cette section, nous proposons une description naïve de l'approche proposée. Avant toute chose, nous introduisons quelques définitions fortes utiles pour la suite. Soit M un POMDP défini par $(S, A, O, T, b_0, \gamma)$, $\{M_k\}_{k=0}^{K}$ un ensemble de sous-POMDPs tel que chaque POMDP M_k soit défini par un n-uplet $(S_k, A, O, T, \Omega, \gamma)$ où : pour tout $k = 0, 1, \cdots, K$, S_k est un sous-espace d'états de S où *couche*. Soit B_k l'ensemble des états de croyance accessibles par simulation dans M_k. Cela est possible en mémorisant à la fois un état $s \in S_k$ où l'agent est effectivement durant la simulation et l'état de croyance $b \in B_k$ de l'agent. À noter que les états de croyance $b \in B_k$ sont des distributions de probabilité sur S et pas exclusivement sur S_k. Ainsi, les trajectoires générées, seront à l'instar de FSVI, des trajectoires de paires, état de l'environnement – état de croyance (s, b). Dans les sous-sections suivantes, nous introduisons deux relations d'ordre des mises à jour : $\prec_1$ défini l'ordre suivant lequel les sous-ensembles $\{B_k\}_{k=0}^{K}$ sont traités ; et $\prec_2$ défini quant à lui l'ordre de mise à jour des états de croyance $b \in B_k$. Nous argumentons que le déterminisme de ces relations d'ordre peut nous permettre d'éviter ou d'atténuer le problème des mises à jour redondantes et/ou inutiles. Encore plus important, nous montrons que leur déterminisme ne requiert pas une grande quantité de ressources.

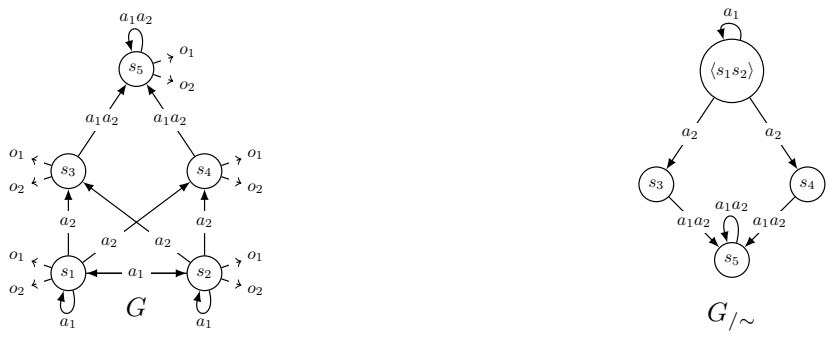


FIG. 1 – Un simple exemple de POMDP, sa représentation graphique G et son graphe de couches $G_{/\sim}$.

Dès lors, le gain dû à l'ordre des mises à jour se fera entièrement sentir dans le temps d'exécution global de notre algorithme lors des expérimentations.

4.1 Ordonner les sous-ensembles d'états de croyance $\{B_k\}_{k=0}^K$

Le problème du déterminisme de l'ordre $\prec_1$ du traitement des sous-ensembles $\{B_k\}_{k=0}^K$, peut être vu comme le problème de l'identification des dépendances causales entre ces sous-ensembles. Notons tout d'abord que les états de croyance et leurs fonctions de valeur sont reliés par une dépendance causale. Afin d'expliquer cette idée, considérons le POMDP M, les sous-ensembles d'états de croyance B_{k+1} et B_k. Si un état de croyance $b' \in B_{k+1}$ est un état de croyance successeur de l'état de croyance $b \in B_k$ pour une paire action – observation (a, o), alors $V(b)$ dépend de $V(b')$. En effet, la fonction de valeur sur b peut être itérativement mise à jour comme suit : $V(b) = \max_a (b \cdot r_a + \gamma \sum_o pr(o|a, b)V(\tau(a, o, b)))$. Pour cette raison, nous voulons mettre à jour b' avant b, *i.e.,* $B_{k+1} \prec_1 B_k$. Il est utile de noter que cette relation est transitive. Toutefois, les POMDPs sont en général cycliques, donc de telles relations sont fréquentes entre états de croyance. Dans ce contexte, nous tournons notre attention vers le problème du déterminisme d'une relation d'ordre $\prec_1$ pour un POMDP cyclique. Une idée employée avec succès en MDPs (Dai & Goldsmith, 2007; Bonet & Geffner, 2003; Abbad & Boustique, 2003) et adaptée ici, consiste à : grouper tous les états qui ont une dépendance causale mutuelle dans une même couche S_k ; ces couches $\{S_k\}_{k=0}^K$ constituent l'espace d'états d'un nouveau POMDP M', acyclique. Dès lors, l'ordre $\prec_1$ des mises à jour des sous-ensembles $\{B_k\}_{k=0}^K$ est donné par l'ordre topologique inverse des états $\{S_k\}_{k=0}^K$ de M'. De façon plus précise, soient deux paires (s, b) et (s', b') associées aux paires (S_k, B_k) et (S_{k+1}, B_{k+1}), respectivement, on a alors $(B_{k+1} \prec_1 B_k)$ si et seulement si $(S_{k+1} \prec_1 S_k)$ et en particulier $(b' \prec_1 b)$ si et seulement si $(s' \prec_1 s)$. Reste cependant à trouver un algorithme du déterminisme des couches $\{S_k\}_{k=0}^K$ qui induisent les sous-ensembles $\{B_k\}_{k=0}^K$. Des travaux en MDPs ont montré que le problème du calcul des couches correspond au problème du calcul des composantes fortement connexes de la représentation graphique du MDP, donc de celle du POMDP où les arrêtes liées aux observations sont omises (Dai & Goldsmith, 2007; Bonet & Geffner, 2003; Abbad & Boustique, 2003). Heureusement, il existe de nombreux algorithmes, y compris celui de Tarjan (1972), qui permettent de trouver en temps linéaire ces couches. La Figure 1 illustre le graphe des couches $G_{/\sim}$, qui est le graphe dont les nœuds sont des couches de G et les arcs $S_k \rightarrow S_{k+1}$ indiquent qu'il existe un état $s \in S_k$ capable de transiter vers un état $s' \in S_{k+1}$. Dans le graphe $G_{/\sim}$ (voir Figure 1), les couches sont $S_0 = \langle s_1 s_2 \rangle$, $S_1 = \langle s_3 \rangle$, $S_2 = \langle s_4 \rangle$, and $S_3 = \langle s_5 \rangle$. Ainsi, l'ordre topologique $\prec_1$ des mises à jour des sous-ensembles $\{B_3, B_2, B_1, B_0\}$ est alors tel que $B_3 \prec_1 B_2 \prec_1 B_1 \prec_1 B_0$.

4.2 Ordonner les mises à jour dans B_k

L'ordre topologique $\prec_1$ présenté ci-dessus est malheureusement incapable d'ordonner les mises à jour des états de croyance d'un même sous-ensemble B_k. En particulier, dans les problèmes de POMDPs où tous les états $s \in S$ ont une dépendance causale les uns avec les autres, il n'y a qu'une seule couche. C'est notamment le cas du banc d'essai `Hallway`. Dans ces situations l'ordre topologique $\prec_1$ est inutile. Pour traiter cette question nous avons recours au principe de chaînage avant ou *projections successives*. Il s'agit d'une procédure qui consiste à : (1) créer par simulation des trajectoires de paires, état de l'environnement – état de croyance, noté (s, b) ; (2) ordonner l'ordre selon lequel les états de croyance $b \in B_k$ sont générés

et mises à jour. Ce dernier point fournira l'ordre topologique $\prec_2$ recherché. Les heuristiques que nous utilisons afin de traverser l'espace des états de croyance sont basées sur la structure du POMDP et l'état réel de l'environnement dans lequel l'agent se trouve lors de la simulation. Pour déterminer un bon ordre de mise à jour des états de croyance dans un même sous-ensemble B_k, nous mémorisons certaines informations supplémentaires. Chaque état s de l'environnement doit se souvenir de ses *prédécesseurs*, *i.e.,* les états qui ont une probabilité non-nulle de transiter vers s. De plus, nous associons à chaque état s une *priorité* $pri(s)$, initialement fixée à la valeur optimale $V^{\star}_{\text{MDP}}(s)$ du MDP sous-jacent. Cette estimation sera utilisée comme une heuristique qui dirige la recherche vers les régions de l'espace d'états qui ont une forte priorité. De plus, l'état de croyance b de la paire (s, b) est mise à jour si et seulement si la priorité associée à l'état s est supérieure à une petite valeur ε préalablement définie, *i.e.,* $pri(s) > \varepsilon$. Notre intuition derrière ce critère est qu'il nous mène vers les états de croyance dont la mise à jour entraînerait une amélioration significative de la fonction de valeur. En effet, la priorité d'un état nous donne une estimation du gain escompté si l'état de croyance associé était mise à jour. À noter qu'un état sous-jacent peut-être associé à plusieurs états de croyance différents. À partir d'une paire, état sous-jacent – état de croyance (s, b), la mise à jour de V fonctionne comme décrit dans l'Algorithme 1. Soit $\hat{p}(s, s')$ la plus grande probabilité d'accéder à s'

Algorithm 1: Sous-routine de MISE À JOUR.

```
UPDATE(s, b, V)
begin
    if pri(s) > psilon then
        Mettre-à-jour l'état de croyance b : V' ← ajout(V, backup(b))
        Réinitialiser la priorité de s pri(s) = 0.
        Calculer la variation Δ = V'(b) − V(b)
        Utiliser Δ pour modifier les priorités des prédécesseurs de s
end
```

en partant de s, alors on a $\hat{p}(s, s') = \max_{(a',o')\in A\times\Omega} T(s'|s, a') \cdot O(o'|s', a')$. Si nous mettons à jour la fonction de valeur V pour une paire, état de l'environnement – état de croyance (s', b), et qu'elle change d'une quantité Δ, les prédécesseurs immédiats de s' en sont informés. Tout état s pour lequel il existe une paire action-observation (a, o) tel que $\hat{p}(s, s') \neq 0$ a sa priorité $pri(s)$ promue à $\Delta \cdot \hat{p}(s, s')$ à moins que cette dernière ait une valeur précédante supérieure.

Afin de déterminer l'état de croyance b^* suivant la paire (s, b), nous déterminons dans un premier temps le triplet état-action-observation (s^*, a^*, o^*) puis en déduisons l'état de croyance b^* comme suit : $b^* = \tau(b, a^*, o^*)$. Pour réaliser ce projet, nous devons déterminer l'état suivant s^*. Cela requiert la distinction entre deux cas de figure :

1. Si (s, b) est associée à une couche qui ne soit pas *résoluble*[1], nous cherchons à générer un chemin aussi court que possible de l'état s vers un état d'une couche dite résoluble. Étant donné G, le problème du déterminisme du plus court chemin entre toutes paires d'états est équivalent au problème du déterminisme du plus court chemin de toutes paires de nœuds $(s, s') \in S^2$ de G. Un graphe qui est tel que chaque arc est étiqueté par un coût défini comme suit :

$$cost(s, s') = \begin{cases} 1 - \hat{p}(s, s') & \text{Si } \hat{p}(s, s') > \varepsilon \\ +\infty & \text{autrement.} \end{cases}$$

   Afin de résoudre un tel problème, nous utilisons l'algorithme de Floyd-Warshall (Cormen *et al.*, 2001) qui requiert une complexité en temps de $O(|S|^3)$ et retourne à la fois la distance la plus courte entre deux états $dist(s, s')$ et l'état suivant le plus court chemin entre ces deux états $next(s, s')$. En conséquence, sachant l'état courant s et un état s_G d'une couche résoluble, l'état suivant est donné par $next(s, s_G)$. Le coût additif dû à l'exécution de l'algorithme de Floyd-Warshall est négligeable comparativement à la complexité de la résolution des POMDPs.

2. Si (s, b) est associé à une couche résoluble S_k, les trajectoires sont générées de façon ordonnée. Nous déterminons dans un premier temps un état s_G qui dispose de la plus grande priorité parmi tous les états de S_k. Puis, nous utilisons les résultats de l'algorithme de Floyd-Warshall et retournons l'état suivant $next(s, s_G)$ comme décrit dans l'algorithme 2. Dans tous les cas, nous utilisons l'état suivant

[1] Une couche est dite résoluble si elle est soit un nœud feuille de la structure du POMDP $G_{/\sim}$ ou une couche qui ne soit dépendante que de couches déjà résolues.

s^* retourné par la routine PICKNEXTSTATE pour déterminer la paire action-observation (a, o). Pour cela, nous avons à notre disposition deux stratégies : (1) la plus simple consiste à sélectionner la paire (a, o) par échantillonnage de $T(s^*|s, \cdot)$ et $O(\cdot|s^*, a)$, respectivement ; (2) la seconde stratégie consiste en la sélection de la paire (a, o) qui maximise la probabilité d'accéder à l'état s^* partant de l'état s, $(a^*, o^*) = \arg\max_{a,o} \tau(b, a, o)[s^*] \cdot pri(s^*)$.

Les trajectoires sont arrêtées lorsqu'elles aboutissent à des états dont les couches ont déjà été résolues, *i.e.,* s^*.TERMINAL $= true$. L'algorithme quant à lui se termine lorsque toutes les couches pertinentes ont été résolues ou alors que la valeur prédéfinie de la *récompense moyenne escomptée* (ADR) est atteinte. Une couche est dite résolue si tous les états relatifs ont leur priorité réduite à zéro. Cela est motivé par l'observation selon laquelle la priorité des états décrit les changements dans la fonction de valeur, donc la résolution d'une couche indique que les états de croyance associés ont convergé pour cette couche.

Algorithm 2: PICKNEXTSTATE sub-routine.

```
PICKNEXTSTATE(s, b)
begin
    pick any state s_G that belongs to a solvable layer
    if ¬s.SOLVABLE then  forall s' ∈ S : s'.SOLVABLE do
        if dist(s, s_G) > dist(s, s') then  s_G ← s'
        else if dist(s, s_G) = dist(s, s') then
            if pri(s_G) < pri(s') then  s_G ← s'
    else  find state s_G with the highest priority
    s* ← next(s, s_G)
end
```

Algorithm 3: TOP for solving POMDPs.

```
TOP(s_0, b_0)
begin
    repeat
        TOPTRIAL(s_0, b_0)
    until V has not converged
end

TOPTRIAL(s, b)
begin
    if ¬s.SOLVED then
        s* ← PICKNEXTSTATE(s, a)
        a* ← PICKACTION(s, s*)
        o* ← PICKNEXTOBSERVATION(s*, a)
        if s*.TERMINAL then  break
        TOPTRIAL(s*, τ(b, a*, o*))
        UPDATE(s, b, V)
end
```

4.3 Analyse de pertinence

Une optimisation peut-être rajoutée afin de restreindre la recherche sur des parties pertinentes du POMDP. En particulier, sachant des informations additionnelles telles que l'état initial et l'état terminal (ceci peut-être aisément étendu au cas de plusieurs états initiaux et terminaux), notre algorithme TOP est capable d'élaguer les parties non pertinentes du problème comme suit :

Théorème 1
Soit M un POMDP, $\{M_k\}_k$ un ensemble de sous-POMDPs induit par les couches $\{S_k\}_k$ de M, où S_K and S_0 sont les couches associées à l'état terminal et l'état initial, respectivement. Ainsi, les affirmations suivantes sont valides :

- *(C1) : Il existe une solution au problème M si et seulement si $S_K \preceq_1 S_0$;*
- *(C2) : Une fonction de valeur de M à ε près de l'optimum peut-être obtenue en résolvant uniquement les sous-POMDPs M_k tels que : $S_K \preceq_1 S_k \preceq_1 S_0$.*

Le théorème ci-dessus défini des conditions de restriction de l'espace de recherche. La preuve de ce théorème s'appuie sur l'observation selon laquelle si la couche S_K succède selon l'ordre topologique $\prec_1$ à la couche S_0, alors il n'existe aucune trajectoire d'états de croyance partant de l'état initial à l'état terminal. En d'autres termes, il n'existe pas de solutions au POMDP M. Le critère de restriction C2 permet la sélection des couches pertinentes pour les informations initiales dont nous disposons. L'explication de ce critère réside dans l'observation selon laquelle les couches S_k en dehors de l'intervalle $S_K \preceq_1 S_k \preceq_1 S_0$ ne peuvent influencer la fonction de valeur retournée.

5 Évaluations empiriques

Nous allons maintenant évaluer la performance de l'algorithme TOP en comparaison aux autres méthodes parmi les plus récentes incluant, HSVI (Smith & Simmons, 2005) ; FSVI ; PVI et PBVI. Nous avons testé ces algorithmes sur plusieurs bancs d'essai de la communauté des POMDPs, y compris : TigerGrid (Cassandra *et al.*, 1994) ; des problèmes de navigations , *e.g.,* Hallway, Hallway2 and Mit (Cassandra *et al.*, 1994) ; ou encore les domaines de Rock Sample (Smith & Simmons, 2004). Nous utilisons le code source de Guy Shani qui implémente tous ces algorithmes, code auquel nous avons ajouté notre algorithme. Les expérimentations ont été effectuées sur la même plateforme Intel Core Duo 1.83GHz CPU avec un processeur 1Gb de mémoire principale.

Nous évaluons ces algorithmes selon les critères suivants : **évaluation de la fonction de valeur–** récompense moyenne escomptée (ADR) : $\frac{\sum_{i=0}^{\#trials}\sum_{j=0}^{\#steps}\gamma^j r_j}{\#trials}$. L'ADR est filtré à l'aide du filtre du premier ordre pour réduire les bruits dans cet estimé. Comme nous nous intéressons à la vitesse de convergence, chaque algorithme a été exécuté jusqu'à ce que sa fonction de valeur atteigne un ADR prédéfini. Nous avons également reporté la valeur de l'état de croyance initial b_0 afin de souligner les effets des différentes stratégies sur la structure de la fonction de valeur. **Temps d'exécution –** nous avons reporté le temps CPU, cependant comme il s'agit de la mise en œuvre d'une implémentation spécifique, nous avons également reporté la quantité d'opérations de mise à jour effectuées. **Mémoire –** nous reportons également le nombre d'hyperplans de la fonction de valeur finale.

De façon similaire à TOP, les algorithmes HSVI et FSVI utilisent tous deux des trajectoires d'états de croyance et ne diffèrent que par la manière dont ces trajectoires sont générées et l'ordre des mises à jour sur les états de croyance. Comme le montre la Table 1, TOP est plus rapide que HSVI, FSVI sur la majeure partie des bancs d'essai et il est très compétitif sur les autres. Bien que nos résultats ne soient que préliminaires à ce jour et ils ne doivent pas être utilisé comme point de comparaison, ils sont pour le moins très encourageants. En comparaison à HSVI et FSVI, TOP réalise le plus petit nombre de mises à jour sur la majeure partie des bancs d'essai. Plus important encore, en n'effectuant des mises à jour que lorsque la priorité des états du MDP sous-jacent est supérieure à ε, TOP réduit considérablement le nombre de mise à jour nécessaires. On remarque, cependant, que si l'ADR de tous les algorithmes était à peu près égal pour tous les bancs d'essai, la valeur de l'état de croyance initial $V(b_0)$ varie considérablement d'un algorithme à un autre, voir par exemple le domaine RockSample $(5, 7)$ à la Table 1. Cela s'explique par la définition des modèles des bancs d'essai, modèles où bien souvent l'état terminal est inexistant. Nous avons également noté que FSVI souffrait du manque d'exploration. En effet, FSVI utilise la Q-valeur du MDP sous-jacent afin de traverser l'espace des états de croyance. Malheureusement, comme la Q-valeur est fixe, FSVI n'explore qu'un petit ensemble des trajectoires possibles, donc il lui est difficile d'améliorer sa fonction de valeur notamment lorsque certaines des trajectoires non explorées peuvent contribuer pour beaucoup à la fonction de valeur. TOP est également plus rapide que les algorithmes PVI et PBVI. Comme nous nous intéressons à la vitesse de convergence, chaque algorithme a été exécuté jusqu'à ce que sa fonction de valeur atteigne l'ADR prédéfini, peu importe le nombre d'états de croyance nécessaires. Tandis que PBVI procède en mettant à jour un ensemble d'états de croyance puis en étendant cet ensemble et ainsi de suite, PVI essaie, 'a coût négligeable, de classer par ordre de priorité ses séquences de mise à jour. Si l'on examine leurs performances par rapport à TOP, nous observons que ces deux algorithmes requièrent plus de temps afin d'atteindre l'ADR prédéfini. Bien que PVI peut effectuer un petit nombre de mises à jour, *e.g.,* Table TigerGrid 1, l'effort requis pour ordonner les séquences d'états de croyance est prohibitif. Les performances de TOP sur l'ensemble des bancs d'essai sont très encourageantes, même si nous notons que les calculs dus à l'algorithme de Floyd-Warshall peuvent entraîner des coûts additifs excessifs. Pour cette raison, nous suggérons l'usage de l'algorithme de Drijkstra, afin de déterminer les plus courts chemins d'un petit nombre de paires d'états. Actuellement, nous examinons cette question, en espérant réduire l'effort

requis par cette sous-routine notamment pour des problèmes encore plus larges.

Méthode	ADR	$V(b_0)$	$\|V\|$	Time (sec)	#mise à jour
TigerGrid					
HSVI	0.68	1.75	1337	535	1695
FSVI	0.66	1.89	1608	380	1275
PBVI	0.66	2.15	472	3749	33101
PVI	0.60	0.69	357	> 3530	480
TOP	0.68	1.05	755	115	640
Hallway					
HSVI	0.51	0.88	258	391	886
FSVI	0.51	0.89	317	440	969
PBVI	0.51	0.75	63	59	1447
PVI	0.51	0.69	185	3272	500
TOP	0.52	0.60	213	56	352
Hallway 2					
HSVI	0.35	0.16	128	126	145
FSVI	0.36	0.24	252	76	204
PBVI	0.35	0.26	128	347	1355
PVI	0.35	0.27	307	> 9565	400
TOP	0.35	0.12	100	21	98
Rock Sample $(4, 4)$					
HSVI	18.036	17.92	173	11	239
FSVI	18.036	13.63	79	7	98
TOP	18.036	13.63	40	3	68
Rock Sample $(5, 5)$					
HSVI	19.237	19.24	50	22	60
FSVI	17.840	13.84	185	200	307
TOP	19.204	17.20	61	37	88
Rock Sample $(5, 7)$					
HSVI	24.202	22.96	208	2754	461
FSVI	22.198	14.97	353	4057	384
TOP	24.086	21.19	317	2791	436
Mit					
HSVI	0.88	0.77	1786	17344	2288
FSVI	0.86	0.69	339	1213	1275
TOP	0.88	0.44	520	820	385

TAB. 1 – Performance measurements.

6 Conclusion

Nous avons introduit un nouvel algorithme de résolution des POMDPs à horizon infini, nommé *planification à base d'ordres topologiques* (TOP). Cet algorithme étudie les propriétés structurelles des POMDPs afin de déterminer la meilleure séquence des mises à jour. Il est basé sur l'idée selon laquelle les différents POMDPs disposent de structures graphiques différentes, or ces dernières déterminent la complexité intrinsèque de la résolution d'un POMDP. Suivant cette observation, TOP parvient à déterminer deux relations d'ordre comme outils pour faire face à la fois à la malédiction de la dimension mais aussi à celle de l'historique. Ces ordres topologiques permettent non seulement d'organiser les mises à jour de la fonction de valeur mais aussi à orienter l'exploration de l'espace des états de croyance, évitant ainsi les mises à jour redondantes et les sous-espaces d'états de croyance non pertinents.

7 Remerciement

Nous remercions Guy Shani pour les discussions constructives sur ce document et pour la mise à notre disposition de son code source.

Références

ABBAD M. & BOUSTIQUE H. (2003). A decomposition algorithm for limiting average markov decision problems. *Oper. Res. Lett.*, **31**(3), 473–476.
BELLMAN R. E. (1957). *Dynamic Programming*. Dover Publications, Incorporated.
BONET B. & GEFFNER H. (2003). Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI*, p. 1233–1238.
CASSANDRA A., LITTMAN M. L. & ZHANG N. L. (1997). Incremental Pruning : A simple, fast, exact method for partially observable Markov decision processes. In *UAI*, p. 54–61.
CASSANDRA A. R., KAELBLING L. P. & LITTMAN M. L. (1994). Acting optimally in partially observable stochastic domains. In *AAAI*, p. 1023–1028.
CORMEN T. H., LEISERSON C. E., RIVEST R. L. & STEIN C. (2001). *Introduction to Algorithms, Second Edition*. The MIT Press.
DAI P. & GOLDSMITH J. (2007). Topological value iteration algorithm for Markov decision processes. In *IJCAI*, p. 1860–1865.
LITTMAN M. L., DEAN T. & KAELBLING L. P. (1995). On the complexity of solving Markov decision problems. In *UAI*, p. 394–402.
LOVEJOY W. S. (1991). Computationally feasible bounds for partially observable markov decison processes. *Operations Research*, **39**, 175–192.
PINEAU J. (2004). *Tractable Planning Under Uncertainty : Exploiting Structure*. PhD thesis, Robotic Institute. Carnegie Mellon University., Pittsburgh.
PINEAU J., GORDON G. & THRUN S. (2003). Point-based value iteration : An anytime algorithm for POMDPs. In *IJCAI*.
POUPART P. & BOUTILIER C. (2004). VDCBPI : an approximate scalable algorithm for large POMDPs. In *NIPs*.
ROY N., GORDON G. & THRUN S. (2005). Finding approximate pomdp solutions through belief compression. *JAIR*, **23**, 1–40.
SHANI G., BRAFMAN R. I. & SHIMONY S. E. (2006). Prioritizing point-based pomdp solvers. In *ECML*, p. 389–400.
SHANI G., BRAFMAN R. I. & SHIMONY S. E. (2007). Forward search value iteration for pomdps. In *IJCAI*, p. 2619–2624.
SMITH T. & SIMMONS R. (2004). Heuristic search value iteration for pomdps. In *UAI*, p. 520–527, Arlington, Virginia, United States.
SMITH T. & SIMMONS R. G. (2005). Point-based POMDP algorithms : Improved analysis and implementation. In *UAI*.
SONDIK E. J. (1978). The optimal control of partially observable markov decision processes over the infinite horizon : Discounted cost. *Operations Research*, **12**, 282–304.
SPAAN M. T. J. & VLASSIS N. (2005). Perseus : Randomized point-based value iteration for POMDPs. *JAIR*, **24**, 195–220.
TARJAN R. (1972). Depth-first search and linear graph algorithms. *SIAM J. Comput.*, **1**(2).
VIRIN Y., SHANI G., SHIMONY S. E. & BRAFMAN R. I. (2007). Scaling up : Solving pomdps through value based clustering. In *AAAI*.
WINGATE D. & SEPPI K. D. (2005). Prioritization methods for accelerating mdp solvers. *J. Mach. Learn. Res.*, **6**, 851–881.

Une nouvelle heuristique pour l'alignement de motifs 2D par programmation dynamique

Émilie Chanoni[1], Thierry Lecroq[2], and Alexandre Pauchet[3]

[1] Université de Rouen, Psy.NCA EA 4306, 76821 Mont-Saint-Aignan Cedex, France,
Emilie.Chanoni@univ-rouen.fr

[2] Université de Rouen, LITIS EA 4108, 76821 Mont-Saint-Aignan Cedex, France,
Thierry.Lecroq@univ-rouen.fr,

[3] INSA-Rouen, LITIS EA 4108, BP 08 - 76801 Saint-Étienne-du-Rouvray, France,
Alexandre.Pauchet@insa-rouen.fr

Résumé : Nous présentons une nouvelle heuristique pour calculer des alignements globaux et locaux de motifs en deux dimensions par programmation dynamique. Cette méthode consiste en une généralisation des formules de récurrence utilisées pour les alignements de séquences. À notre connaissance, cette étude est la première qui considère le calcul d'alignements locaux de motifs 2D qui plus est avec des scores quelconques. Nous avons évalué cette méthode sur la recherche de motifs répétés sur des annotations de dialogues entre parents et enfants de quatre ans lors de la narration de deux histoires enfantines. Nous avons pu mettre en évidence un motif commun à plusieurs dialogues.

1 Introduction

La programmation dynamique est une technique utilisée dans de multiples applications, voir par exemple Lemaître *et al.* (2002); Hansen *et al.* (2004); Szer & Charpillet (2006); Sutton (1990). Les mots (séquences de symboles appartenant à un alphabet Σ) peuvent être facilement comparés à l'aide de cette technique. Des applications de ce problème incluent la bioinformatique, le traitement automatique de la langue naturelle ou bien le traitement de la parole. Les arbres peuvent aussi être comparés en utilisant des techniques similaires. Le lecteur peut se référer à Sankoff & Kruskal (1999) pour de plus amples détails. Il y a un grand nombre de travaux considérant la recherche exacte ou approchée en deux dimensions : étant donnés un motif X et un texte Y, tous les deux en deux dimensions, trouver les occurrences de X dans Y (voir Amir *et al.* (1994); Kärkkäinen & Ukkonen (1994); Amir & Farach (1997)). Il y a eu des efforts pour indexer des matrices en utilisant des arbres de suffixes ou des tables de suffixes (Giancarlo & Grossi, 1997; Kim *et al.*, 2003; Na *et al.*, 2007) mais ces index permettent de trouver des répétitions exactes et la généralisation pour trouver des motifs approchés est loin d'être immédiate. Très peu d'attention a été portée sur l'alignement de matrices à deux dimensions bien que les applications de ce problème incluent des domaines fondamentaux comme la fouille de donnés ou le traitement des images. Krithivasan & Sitalakshmi (1987) considère des motifs en deux dimensions de même taille alors que Baeza-Yates (1998) considère seulement l'alignement global de deux motifs en

deux dimensions. À notre connaissance, le présent article constitue la première tentative pour formaliser le calcul d'alignements locaux de deux motifs en deux dimensions par programmation dynamique. Récemment Arslan (2007) a considéré le problème de la recherche de plus long sous motifs communs en dimensions $d > 2$ mais en utilisant des techniques différentes.

Nous donnons des formules de récurrence pour calculer des alignements globaux et locaux de motifs en deux dimensions de taille respective M et N en temps et espace $O(M \times N)$. Pour cela nous devons précalculer les similarités entre tous les préfixes de toutes les lignes et tous les préfixes de toutes les colonnes des deux motifs. Cela peut également être effectué en temps et espace $O(M \times N)$.

Une première évaluation de cette méthode sur la recherche de motifs répétés sur des annotations de dialogues entre parents et enfants de quatre ans lors de la narration de deux histoires enfantines a été réalisée. Elle a déjà permis d'identifier un motif significatif dans ces dialogues.

Le reste de l'article est organisé comme suit : la section 2 rappelle les techniques de programmation dynamique pour l'alignement de séquences ; la section 3 présente la nouvelle méthode de programmation dynamique pour l'alignement de motifs en deux dimensions ; la section 4 montre l'évaluation réalisée, enfin la section 5 donne nos conclusions et des perspectives.

2 L'alignement de séquences

Les alignements sont généralement utilisés pour comparer les séquences. Ils sont largement répandus en bioinformatique. Ils constituent un moyen pour visualiser la ressemblance entre les séquences. Ils sont basés sur des notions de distance ou de similarité. Ils sont usuellement calculés par programmation dynamique. On considère deux types d'alignements de deux séquences x et y : les alignements globaux (qui considèrent la totalité des deux séquences) et les alignements locaux (qui permettent de détecter le segment de x le plus similaire à un segment de y).

Exemple

A	**C**	**G**	–	–	**A**
A	**T**	**G**	**C**	**T**	**A**

est un alignement global de **ACGA** et **ATGCTA**.

Une solution peut aussi être donnée sous forme de script d'édition comme suit :

Opération	séquence résultante
substitution de **A** par **A**	**A**
substitution de **C** par **T**	**AT**
substitution de **G** par **G**	**ATG**
insertion de **C**	**ATGC**
insertion de **T**	**ATGCT**
substitution de **A** par **A**	**ATGCTA**

2.1 Alignement global

Un alignement global de deux séquences x et y peut être obtenu en calculant la distance entre x et y. La notion de distance entre deux séquences est largement utilisée pour comparer des fichiers. La commande **diff** d'UNIX implante un algorithme basé sur cette notion où chaque ligne des fichiers est considérée comme un symbole. La sortie d'une comparaison des deux fichiers effectuée par **diff** est le nombre minimum

d'opérations d'édition (substitution, insertion ou suppression) pour transformer le premier fichier en le deuxième. Cette commande réalise un alignement global entre les deux fichiers.

On définit la distance d'édition entre deux séquences x et y comme étant le nombre minimum d'opérations d'édition élémentaires pour transformer x en y. Les opérations d'édition élémentaires sont : substitution d'un symbole de x par un symbole de y, suppression d'un symbole de x et insertion d'un symbole de y. Il est également possible de calculer des alignements globaux en utilisant des scores de similarités plutôt qu'une distance. Un score est associé à chaque opération d'édition élémentaire. Pour $a, b \in \Sigma$: *Sub*(a, b) est le score de la substitution du symbole a par le symbole b, *Del*(a) est le score de la suppression du symbole a, *Ins*(a) est le score de l'insertion du symbole a. Pour deux symboles a et b, une valeur positive de *Sub*(a, b) signifie que les deux symboles sont proches (ou similaires), et une valeur négative de *Sub*(a, b) signifie que les deux symboles sont éloignés (ou dissimilaires). Nous pouvons maintenant définir le score d'édition généralisé entre deux séquences x et y par $d(x, y) = \max\{\text{score de } \gamma \mid \gamma \in \Gamma_{x,y}\}$ où $\Gamma_{x,y}$ est l'ensemble de toutes les séquences d'opérations d'édition qui transforment x en y, et le score d'un élément $\gamma \in \Gamma_{x,y}$ est la somme des scores de ses opérations d'édition élémentaires.

Pour calculer $d(x, y)$ pour deux séquences x et y de longueur respective m et n, on utilise une table à deux dimensions t à $m + 1$ lignes et $n + 1$ colonnes telle que $t[i, j] = d(x[0\,..\,i], y[0\,..\,j])$ pour $i = 0, \ldots, m - 1$ and $j = 0, \ldots, n - 1$. Il s'ensuit que $d(x, y) = t[m - 1, n - 1]$.

Les valeurs de la table t peuvent être calculées en suivant les formules de récurrence pour $i = 0, 1, \ldots, m - 1$ et $j = 0, 1, \ldots, n - 1$:

$$t[-1, -1] = 0, t[i, -1] = t[i - 1, -1] + \mathit{Del}(x[i]), t[-1, j] = t[-1, j - 1] + \mathit{Ins}(y[j]),$$

$$t[i, j] = \max \begin{cases} t[i - 1, j - 1] + \mathit{Sub}(x[i], y[j]) \,, \\ t[i - 1, j] + \mathit{Del}(x[i]) \,, \\ t[i, j - 1] + \mathit{Ins}(y[j]) \,. \end{cases}$$

La valeur à la position (i, j) de la table t ne dépend que des valeurs aux trois positions voisines $(i - 1, j - 1)$, $(i - 1, j)$ et $(i, j - 1)$ (voir Crochemore *et al.* (2007)).

Une application directe de la formule de récurrence ci-dessus donne un algorithme en temps exponentiel pour calculer $t[m - 1, n - 1]$. Cependant la totalité de la table t peut être calculée en temps quadratique, par une technique connue sous le nom de « programmation dynamique ». C'est une technique générale qui peut être utilisée pour résoudre différents types d'alignements.

Un alignement optimal (de score maximal) peut alors être produit. Cela consiste à effectuer un tracé arrière du calcul des valeurs de la table t à partir de la position $(m - 1, n - 1)$ jusqu'à la position $(-1, -1)$.

2.2 Alignement local

Calculer un alignement local de deux séquences x et y consiste à trouver le segment de x qui est le plus similaire à un segment de y.

Le score local d'édition de deux séquences x et y est défini par :
$s(x, y) =$ similarité maximale entre un segment de x et un segment de y.

Pour calculer $s(x, y)$ pour deux séquences x et y de longueur respective m et n, nous utilisons une table à deux dimensions t_S à $m + 1$ lignes et $n + 1$ colonnes telle

que $t_S[i,j] = \max\{s(x[\ell\,..\,i], y[k\,..\,j]) \mid 0 \leq \ell \leq i \text{ et } 0 \leq k \leq j\} \cup \{0\}$, pour $i = 0, \ldots, m-1$ et $j = 0, \ldots, n-1$. Donc $s(x,y) =$ valeur maximale dans t_S.

Les valeurs de la table t_S peuvent être calculées avec les formules de récurrence suivantes : pour $i = 0, 1, \ldots, m-1$ et $j = 0, 1, \ldots, n-1$:

$$t_S[-1,-1] = t_S[i,-1] = t_S[-1,j] = 0$$

$$t_S[i,j] = \max \begin{cases} t_S[i-1,j-1] + \mathit{Sub}(x[i], y[j])\,, \\ t_S[i-1,j] + \mathit{Del}(x[i])\,, \\ t_S[i,j-1] + \mathit{Ins}(y[j])\,, \\ 0\,. \end{cases}$$

Calculer les valeurs de t_S pour un alignement local de x et y peut être fait en temps et espace $O(mn)$. Retrouver un alignement local peut être fait d'une manière similaire au cas des alignements globaux à ceci près que le tracé arrière doit commencer à une position d'une valeur maximale dans t_S plutôt qu'à la position $(m-1, n-1)$.

3 Alignement de motifs 2D

Considérons maintenant deux motifs rectangulaires $X = X[0\,..\,m_1-1, 0\,..\,n_1-1]$ et $Y = Y[0\,..\,m_2-1, 0\,..\,n_2-1]$, de taille respective $M = m_1 \times n_1$ et $N = m_2 \times n_2$. Chaque élément $X[i,j]$ avec $0 \leq i \leq m_1 - 1$ et $0 \leq j \leq n_1 - 1$ et $Y[k,\ell]$ avec $0 \leq k \leq m_2 - 1$ et $0 \leq \ell \leq n_2 - 1$ appartient à l'alphabet Σ.

Nous voulons maintenant calculer le score maximum des opérations d'insertion, de suppression ou de substitution de symboles individuels pour transformer X en Y. Les symboles peuvent être insérés, supprimés ou substitués séparément ou par portion de lignes ou par portion de colonnes.

3.1 Alignement global

Aligner X et Y en utilisant la programmation dynamique consiste à généraliser les formules de récurrence utilisées pour l'alignement de séquences.

Pour cela nous avons besoin de quatre tables à quatre dimensions D_R, D_C, I_R et I_C définies comme suit pour $0 \leq i \leq m_1 - 1$, $0 \leq j \leq n_1 - 1$, $0 \leq i \leq m_2 - 1$ et $0 \leq j \leq n_2 - 1$:

$$D_R[i,j] = \sum_{p=0}^{j} \mathit{Del}(X[i,p]) \qquad \text{et} \qquad D_C[i,j] = \sum_{p=0}^{i} \mathit{Del}(X[p,j])$$

$$I_R[i,j] = \sum_{p=0}^{j} \mathit{Ins}(Y[i,p]) \qquad \text{et} \qquad I_C[i,j] = \sum_{p=0}^{i} \mathit{Ins}(Y[p,j]).$$

En d'autres termes, $D_R[i,j]$ est le score de la suppression du préfixe de longueur $j+1$ de la ligne i de X, $D_C[i,j]$ est le score de la suppression du préfixe de longueur $i+1$ de la colonne j de X, $I_R[i,j]$ est le score de l'insertion du préfixe de longueur $j+1$ de la ligne i de Y et $I_C[i,j]$ est le score de l'insertion du préfixe de longueur $i+1$ de la ligne j de Y. Les tables D_R et D_C peuvent être calculées en temps et espace $O(m_1 \times n_1)$. Les tables I_R et I_C peuvent être calculées en temps et espace $O(m_2 \times n_2)$.

Nous allons aussi utiliser deux tables à quatre dimensions R et C de taille $m_1 \times n_1 \times m_2 \times n_2$ definies comme suit : $R[i,j,k,\ell] = d(X[i,0\,..\,j], Y[k,0\,..\,\ell])$, et

$C[i,j,k,\ell] = d(X[0\,..\,i,j], Y[0\,..\,k,\ell])$. En fait $R[i,j,k,\ell]$ contient la distance entre le préfixe de longueur $j+1$ de la ligne i de X et le préfixe de longueur $\ell+1$ de la ligne k de Y. De manière similaire $C[i,j,k,\ell]$ contient la distance entre le préfixe de longueur $i+1$ de la colonne j de X et le préfixe de longueur $k+1$ de la colonne ℓ de Y. Les deux tables R et C peuvent être calculées en temps et espace $O(m_1 \times n_1 \times m_2 \times n_2)$.

Nous utilisons alors une table à quatre dimensions T de taille $(m_1+1)\times(n_1+1)\times(m_2+1)\times(n_2+1)$ définie comme suit : $T[i,j,k,\ell] = \max\{\text{score de } \gamma \mid \gamma \in \Gamma_{X,Y}\}$ où $\Gamma_{X,Y}$ est l'ensemble de toutes les séquences d'opérations d'édition qui transforment X en Y, et le score d'un élément $\gamma \in \Gamma_{x,y}$ est la somme des scores de ses opérations d'édition élémentaires.

Les valeurs de la table T peuvent être calculées comme suit pour $0 \le i \le m_1 - 1$, $0 \le j \le n_1 - 1$, $0 \le k \le m_2 - 1$ et $0 \le \ell \le n_2 - 1$ (voir Fig. 1) :

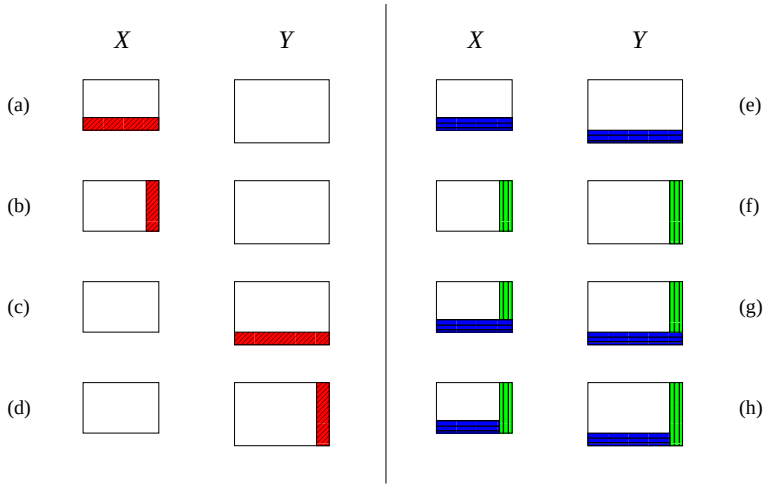


FIG. 1 – (a) : suppression de $X[i,0\,..\,j]$; (b) : suppression de $X[0\,..\,i,j]$; (c) : insertion de $Y[k,0\,..\,\ell]$; (d) : insertion de $Y[0\,..\,k,\ell]$; (e) : substitution de $X[i,0\,..\,j]$ par $Y[k,0\,..\,\ell]$; (f) : substitution de $X[0\,..\,i,j]$ par $Y[0\,..\,k,\ell]$; (g) : substitution de $X[i,0\,..\,j]$ par $Y[k,0\,..\,\ell]$ et substitution de $X[0\,..\,i-1,j]$ par $Y[0\,..\,k-1,\ell]$; (h) : substitution de $X[i,0\,..\,j-1]$ par $Y[k,0\,..\,\ell-1]$ et substitution de $X[0\,..\,i,j]$ par $Y[0\,..\,k,\ell]$.

$$T[i,j,k,\ell] = \max \begin{cases} T[i-1,j,k,\ell] + D_R[X[i,0\,..\,j]] \\ T[i,j-1,k,\ell] + D_C[X[0\,..\,i,j]] \\ T[i,j,k-1,\ell] + I_R[Y[k,0\,..\,\ell]] \\ T[i,j,k,\ell-1] + I_C[Y[0\,..\,k,\ell]] \\ T[i-1,j,k-1,\ell] + R[i,j,k,\ell] \\ T[i,j-1,k,\ell-1] + C[i,j,k,\ell] \\ T[i-1,j-1,k-1,\ell-1] + C[i-1,j,k-1,\ell] + R[i,j,k,\ell] \\ T[i-1,j-1,k-1,\ell-1] + C[i,j,k,\ell] + R[i,j-1,k,\ell-1] \end{cases} \tag{1}$$

Nous avons les initialisations marginales suivantes : $T[-1,j,k,\ell] = T[i,-1,k,\ell] = (k+1)\times(\ell+1)$ et $T[i,j,-1,\ell] = T[i,j,k,-1] = (i+1)\times(j+1)$ pour $0 \le i \le m_1-1$, $0 \le j \le n_1 - 1$, $0 \le k \le m_2 - 1$ et $0 \le \ell \le n_2 - 1$.

Ensuite un tracé arrière peut être effectué à partir de $T[m_1, n_1, m_2, n_2]$ comme dans le cas de l'alignement de séquences.

L'alignement global de deux motifs rectangulaires de taille respective $M = m_1 \times n_1$ et $N = m_2 \times n_2$ peut être effectué en temps et espace $O(M \times N)$.

Exemple

Avec les motifs suivants

X	0	1	2
0	**A**	**B**	**C**
1	**D**	**E**	**F**
2	**G**	**H**	**I**
3	**J**	**K**	**L**

et

Y	0	1
0	**E**	**C**
1	**H**	**I**
2	**K**	**L**

et en considérant les scores suivants pour les opérations d'édition (*Ins*$(a) =$ *Del*$(a) = -1$, *Sub*$(a, a) = 2$ et *Sub*$(a, b) = -1$ pour $a, b \in \Sigma$ tels que $a \neq b$), un script possible en remontant depuis $T[3, 2, 2, 1] = 6$ est :

Opération	Score
Substitution de $X[0\,..\,3, 2]$ par $Y[0\,..\,2, 1]$	5
Substitution de $X[0\,..\,3, 1]$ par $Y[0\,..\,2, 0]$	5
Suppression de $X[0\,..\,3, 0]$	-4

3.2 Alignement local

Pour calculer un alignement local entre deux motifs rectangulaires X et Y on utilise deux tables à quatre dimensions R_S et C_S de taille $m_1 \times n_1 \times m_2 \times n_2$ définies comme suit :

$R_S[i, j, k, \ell] = s(X[i, 0\,..\,j], Y[k, 0\,..\,\ell])$ et $C_S[i, j, k, \ell] = s(X[0\,..\,i, j], Y[0\,..\,k, \ell])$.

En fait, $R_S[i, j, k, \ell]$ contient la similarité maximale entre un suffixe de $X[i, 0\,..\,j]$ et un suffixe de $Y[k, 0\,..\,\ell]$. De manière similaire $C_S[i, j, k, \ell]$ contient la similarité maximale entre un suffixe de $X[0\,..\,i, j]$ et un suffixe de $Y[0\,..\,k, \ell]$. Ces deux tables peuvent être calculées en utilisant la formule de récurrence habituelle pour les séquences en temps et espace $O(m_1 \times n_1 \times m_2 \times n_2)$.

Ensuite on utilise une table à quatre dimensions T_S de taille $(m_1 + 1) \times (n_1 + 1) \times (m_2 + 1) \times (n_2 + 1)$ définie comme suit :

$$\begin{aligned} T_S[i, j, k, \ell] = \max \quad & \{s(X[i'\,..\,i, j'\,..\,j], Y[k'\,..\,k, \ell'\,..\,\ell]) \mid \\ & 0 \leq i' \leq i, 0 \leq j' \leq j, 0 \leq k' \leq k \text{ et } 0 \leq \ell' \leq \ell\} \cup \{0\}. \end{aligned}$$

On note : $r = R_S[i, j, k, \ell]$, $c = C_S[i, j, k, \ell]$, $r' = R_S[i, j-1, k, \ell-1]$, $c' = C_S[i-1, j, k-1, \ell]$ et $q =$ *Del*$(X[i, j]) +$ *Ins*$(Y[k, \ell])$.

Les valeurs de la table T_S peuvent être calculées à l'aide de l'équation ci-dessous.

$$T_S[i, j, k, \ell] = \max \begin{cases} T_S[i-1, j, k, \ell] + \textit{Del}(X[i, j]) \\ T_S[i, j-1, k, \ell] + \textit{Del}(X[i, j]) \\ T_S[i, j, k-1, \ell] + \textit{Ins}(Y[k, \ell]) \\ T_S[i, j, k, \ell-1] + \textit{Ins}(Y[k, \ell]) \\ T_S[i-1, j, k-1, \ell] + (r \text{ si } r \neq 0 \text{ sinon } q) \\ T_S[i, j-1, k, \ell-1] + (c \text{ si } c \neq 0 \text{ sinon } q) \\ T_S[i-1, j-1, k-1, \ell-1] + (c' + r \text{ si } c', r \neq 0 \text{ sinon } q) \\ T_S[i-1, j-1, k-1, \ell-1] + (c + r' \text{ si } c, r' \neq 0 \text{ sinon } q) \\ 0 \end{cases} \tag{2}$$

pour $0 \leq i \leq m_1 - 1$, $0 \leq j \leq n_1 - 1$, $0 \leq k \leq m_2 - 1$ et $0 \leq \ell \leq n_2 - 1$.

Nous avons les initialisations marginales suivantes :

$$T_S[-1, j, k, \ell] = T_S[i, -1, k, \ell] = T_S[i, j, -1, \ell] = T_S[i, j, k, -1] = 0$$

pour $0 \leq i \leq m_1 - 1$, $0 \leq j \leq n_1 - 1$, $0 \leq k \leq m_2 - 1$ et $0 \leq \ell \leq n_2 - 1$.

Les valeurs de la table T_S de taille $(m_1 + 1) \times (n_1 + 1) \times (m_2 + 1) \times (n_2 + 1)$ permettant de détecter des motifs similaires peuvent être calculées en temps et espace $O(m_1 \times n_1 \times m_2 \times n_2)$.

La procédure de tracé arrière doit commencer à la position contenant une valeur maximale plutôt qu'à la position $(m_1 - 1, n_1 - 1, m_2 - 1, n_2 - 1)$.

Exemple

Avec le système de score suivant : *Sub*$(a, a) = 1$, *Sub*$(a, b) = -1$ et *Ins*$(a) =$ *Del*$(a) = -1$, pour $a, b \in \Sigma$ tels que $a \neq b$, la Fig. 2 montre sur la gauche les deux motifs à aligner et sur la droite le meilleur alignement local où le symbole **c** sur la troisième ligne du second motif est inséré, tandis que les symboles **ghi** de la troisième ligne du premier motif sont substitués par les symboles **aba** de la quatrième ligne du second motif.

```
A b C J e f g T i j 5 l m n o
B b C d E f g h i j k l m n o
B b C J L f g h i j k l m n o
A b C d E f g h i j k l m n o
```

```
A b C J e f g T i
B b C d E f g h i
B b C J L f g h i
A b C d E f g h i j k l m n
```

```
A b C J e f g T i j 5 l m n
B b C d E f g h i , ; : ! ?
x y z t u v w a c e p q r s
B b C J L f a b a b a c a b
A b C d E f g h i j k l m n
a j g j h a j h i j a j j j
```

```
A b C J e f g T i
B b C d E f g h i
                c
B b C J L f a b a
A b C d E f g h i j k l m n
```

FIG. 2 – Alignement local de motifs 2D.

4 Applications

L'étude de fichiers log et de corpus de dialogues annotés vise la plupart du temps à repérer des comportements récurrents significatifs, afin de concevoir des outils de contrôle du système ou d'aide à la décision. Ces comportements apparaissent alors sous la forme de motifs répétés, représentés par des automates (Winograd & Flores, 1986; Bouzouba & Moulin, 2001), d'automates temporisés (Pauchet, 2006), de réseaux de Petri (Mazouzi, 2001), de diagrammes de séquences (FIPA, 2002), *etc.* Le point commun de toutes ces représentations étant qu'elles s'appuient sur des motifs linéaires.

Dans le cas particulier de corpus de dialogues dont les annotations visent à caractériser un processus émotionnel, chaque énoncé doit être annoté par une série de termes. Il en résulte une recherche de répétitions non plus linéaires, mais en deux dimensions, rendant impossible les approches classiques des études de corpus. L'approche, présentée ici, de recherche d'alignements de motifs 2D répond tout à fait à cette problématique. Il s'agit d'une démarche en trois étapes :

1. identification d'alignements de motifs en 2D grâce à l'algorithme présenté précédemment ;

2. évaluation de la pertinence sémantique des motifs repérés (expertise) ;
3. évaluation statistique de ces motifs.

L'algorithme a été testé sur des dialogues entre parents et enfants de quatre ans lors de la narration de deux histoires enfantines (A et B). Chaque énoncé retranscrit et codé selon la grille de Chanoni (2004) correspond à une ligne. Cette grille a été utilisée dans le cadre d'une étude psychologique portant sur les interprétations mentalistes des comportements d'un personnage. L'objectif étant de repérer les caractéristiques dialogiques, sémantiques et pragmatiques du discours que proposent les adultes pour expliquer le comportement d'un personnage soumis à une fausse croyance (Wimmer & Perner, 1983). Cette grille est composée de cinq colonnes, chaque colonne comporte entre 2 et 7 codages (6, 3, 7, 2, 2) : les indices relevés dans la première colonne concernent la nature de l'énoncé. Il peut s'agir d'une affirmation (a ou A), d'une question (q ou Q), d'une demande d'attention sur l'histoire (AH), d'une demande d'attention générale (AG). La deuxième colonne concerne la référenciation de l'énoncé. L'énoncé peut se référer au personnage (P), à l'interlocuteur (HR) ou au locuteur (AR). La troisième colonne relève les états mentaux. Les partenaires du dialogue peuvent décrire l'émotion (E), la volition (V), la cognition observable ou non observable (CO ou CNO), l'épistémie (EP), l'hypothèse (HY) et la surprise (S). Ici la surprise est distinguée des émotions pour ses liens avec la croyance dont elle est issue. Les deux dernières colonnes sont consacrées aux justifications par cause / conséquence (C) ou par opposition (O) qui peuvent être appliquées soit pour expliquer l'histoire (CH), soit pour expliquer une situation par l'évocation d'un contexte personnel (CP). Une matrice de substitution a été spécifiquement construite pour cette application.

Après avoir réalisé les alignements de toutes les paires de dialogues et les alignements intra-dialogues, nous nous sommes focalisés sur trois dialogues. Deux des dialogues concerne l'histoire A et un autre l'histoire B, les partenaires de dialogues sont tous différents. Cette technique nous a permis de repérer des sous-motifs approchés (ou flous) se répétant à l'intérieur d'un même dialogue ainsi qu'au sein des deux autres. Sur des séries de 12, 13 et 16 lignes (selon les localisations) nous observons la répétition d'un motif qui se décompose en deux parties : le premier ensemble est composé d'une série de lignes (de 9 à 14) qui comportent presque des indices relatant la présence d'un état mental et d'une référenciation. Ce premier ensemble est suivi d'une série de deux lignes qui comporte chacune une justification associée à un état mental.

L'intérêt de ce sous-motif réside dans le fait qu'il semble caractériser à la fois une organisation dialogique et un contenu : fait délicat voire impossible à observer « à l'œil nu ». Pour cet exemple, il apparaît que les parents prennent soin de préparer la justification « mentaliste » de la situation. En effet, dans un premier temps, ils semblent décrire et préciser les états mentaux en jeu, puis dans un deuxième temps, ils précisent les liens de cause, de conséquence ou d'opposition qui expliquent les états mentaux du personnage et ses comportements.

À terme, tous les motifs détectés seront répertoriés et recherchés dans tous les dialogues. Ainsi pour chaque symbole de tous les dialogues, il sera possible de savoir dans quel motif il apparaît et quels sont les dialogues (avec les positions) dans lesquels ces motifs apparaissent. Le but étant d'aider un expert à détecter des motifs significatifs.

5 Conclusion et perspectives

Nous avons présenté une méthode très générique pour aligner des motifs en deux dimensions par programmation dynamique en temps et espace quadratique. Cette métho-

de permet de calculer des alignements globaux et locaux. Une première difficulté consiste à visualiser les alignements ainsi calculés. Beaucoup de questions peuvent être posées. Est-il possible d'aligner des motifs 2D en temps sous-quadratique comme c'est le cas pour les séquences ? (voir Crochemore *et al.* (2003)). Est-il possible de développer une heuristique similaire à BLAST (Altschul *et al.*, 1990) pour rapidement aligner un motif 2D contre une banque de motifs 2D ? D'autre part, il est nécessaire d'accompagner la méthode proposée d'une validation statistique pour estimer la pertinence des résultats trouvés.

Au-delà de l'intérêt scientifique et théorique, cette méthode peut à terme, non seulement fournir un outil d'aide au diagnostic dans le cadre de situations dialogiques asymétriques mais pourrait plus généralement fournir un outil à la formation à l'entretien.

En effet, lors d'une interaction des émotions non perceptibles par des termes explicites atteignent tout de même la subjectivité de l'interlocuteur, ces émotions sont indispensables à prendre en compte pour une bonne qualité de dialogue, elles le sont d'autant plus lors d'interactions visant à un diagnostic.

Imaginons un jeune enfant qui vient consulter pour une douleur à l'abdomen. L'enfant et le médecin n'ont ni le même niveau de langage, ni le même niveau de représentation du corps et de la douleur. Pourtant le diagnostic ne se fera que sur la base des propos que l'enfant aura eu à propos de sa douleur. Cette méthode en complément d'un entretien semi dirigé pourrait permettre de saisir des informations non saisissables d'emblée et pourtant nécessaires au diagnostic.

Par ailleurs, cette méthode accompagnée d'une expertise psychologique permet de relever les techniques ou les patterns (dialogiques ou non) efficaces. Ainsi, la formation aux techniques d'entretien pourrait s'en trouver améliorée. Nous pourrions penser qu'un jour cette méthode pourrait assister en temps réel l'apprenant dans sa formation à l'entretien.

Références

ALTSCHUL S. F., GISH W., MILLER W., MYERS E. W. & LIPMAN D. J. (1990). A basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

AMIR A., BENSON G. & FARACH M. (1994). An alphabet independent approach to 2-dimensional pattern matching. *SIAM J. Comput.*, **23**(2), 313–323.

AMIR A. & FARACH M. (1997). Two dimensional matching. In A. APOSTOLICO & Z. GALIL, Eds., *Pattern Matching Algorithms*, chapter 9, p. 267–292. Oxford University Press.

ARSLAN A. N. (2007). A largest common d-dimensional subsequence of two d-dimensional strings. In *FCT*, volume 4639 of *LNCS*, p. 40–51.

BAEZA-YATES R. A. (1998). Similarity in two-dimensional strings. In *COCOON*, volume 1449 of *LNCS*, p. 319–328.

BOUZOUBA K. & MOULIN B. (2001). Les interactions distribuées : Approche basée sur le point de vue personnel des agents. In *MFI*, p. 15–30.

CHANONI E. (2004). *Rôle du langage dans le développement de la théorie de lesprit chez les enfants de 3 à 5 ans : contexte verbal et contexte narratif*. PhD thesis, Presse Universitaire de Lille.

CROCHEMORE M., HANCART C. & LECROQ T. (2007). *Algorithms on strings*. Cambridge University Press.

CROCHEMORE M., LANDAU G. M. & ZIV-UKELSON M. (2003). A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM J. Comput.*, **32**(6), 1654–1673.

GIANCARLO R. & GROSSI R. (1997). Suffix tree data structures for matrices. In A. APOSTOLICO & Z. GALIL, Eds., *Pattern matching algorithms*, chapter 10, p. 293–340. Oxford University Press.

HANSEN E., BERNSTEIN D. & ZILBERSTEIN S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on AI*, p. 709–715.

KÄRKKÄINEN J. & UKKONEN E. (1994). Two and higher dimensional pattern matching in optimal expected time. In *SODA*, p. 715–723.

KIM D. K., KIM Y. A. & PARK K. (2003). Generalizations of suffix arrays to multidimensional matrices. *Theoret. Comput. Sci.*, **302**(1–3), 223–238.

KRITHIVASAN K. & SITALAKSHMI R. (1987). Efficient two-dimensional pattern matching in the presence of errors. *Inf. Sci.*, **43**, 169–184.

LEMAÎTRE M., VERFAILLIE G., JOUHAUD F., LACHIVER J.-M. & BATAILLE N. (2002). Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, **6**(5), 367–381.

MAZOUZI H. (2001). *Ingénierie des protocoles d'interaction : des systèmes distribués aux systèmes multi-agents*. PhD thesis, Université Paris IX.

NA J. C., GIANCARLO R. & PARK K. (2007). On-line construction of two-dimensional suffix trees in $O(n^2 \log n)$ time. *Algorithmica*, **48**(2), 173–186.

PAUCHET A. (2006). *Modélisation cognitive d'interactions humaines dans une cadre de planification multi-agents*. PhD thesis, Université Paris Nord.

SANKOFF D. & KRUSKAL J. B. (1999). *Time warps, string edits, and macromolecules : the theory and practice of sequence comparison*. Cambridge University Press. 2nd edition.

SUTTON R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Conference on Machine Learning*, p. 216–224.

SZER D. & CHARPILLET F. (2006). Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the 21st National Conference on AI*.

FIPA (2002). FIPA *Request Interaction Protocol Specification*. Rapport interne, Foundation for Intelligent Physical Agents. Avail. at `http://www.fipa.org`.

WIMMER H. & PERNER J. (1983). Beliefs about beliefs : representation and constraining function of the wrong beliefs in young childrens understandig of deception. *Cognition*, **13**, 103–128.

WINOGRAD T. & FLORES F. (1986). *Understanding Computers and Cognition : A New Foundation for Design*. Ablex.

Apprentissage par renforcement hiérarchique dans les MDP factorisés *

Olga Kozlova[1,2], Olivier Sigaud[1] et Christophe Meyer[3]

[1] Institut des Systèmes Intelligents et de Robotique
Université Pierre et Marie Curie - Paris 6, CNRS FRE 2507
4 place Jussieu, F75252 Paris Cedex 05
Olivier.Sigaud@lip6.fr et http://www.isir.fr/

[2] Thales Security Solutions & Services, Simulation
1 rue du Général de Gaulle, Osny BP 226
F95523 Cergy Pontoise Cedex
Olga.Kozlova@thalesgroup.com

[3] Thales Security Solutions & Services, ThereSIS Research and Innovation Office
Route départementale 128
F91767 Palaiseau Cedex
Christophe.Meyer@thalesgroup.com

Résumé : Nous proposons dans cet article une méthode qui permet d'apprendre par renforcement à résoudre un problème markovien de grande taille dont on ne connaît pas a priori la structure en combinant les capacités d'abstraction hiérarchique des Processus Décisionnels Semi-Markoviens et les capacités de factorisation des Processus Décisionnels de Markov factorisés. Nous validons notre approche sur le problème classique du taxi.
Mots-clés : Apprentissage par renforcement, modèles factorisés, options.

1 Introduction

Les Processus Décisionnels de Markov (MDP) constituent un cadre fondamental pour modéliser des problèmes de planification et d'apprentissage par renforcement dans l'incertain. Cependant, les algorithmes exacts conçus dans ce cadre sont inaptes à traiter des problèmes de grande taille à cause de la « malédiction de la dimensionalité » (Sutton & Barto, 1998). Les MDP factorisés (FMDP) (Boutilier *et al.*, 1995) permettent de représenter des problèmes structurés de grande taille de façon compacte. Dans cette approche, un état est décrit implicitement par l'assignation de valeurs à un ensemble de variables aléatoires. Les réseaux bayésiens dynamiques (DBN) (Dean & Kanazawa, 1989) permettent alors d'exploiter les dépendances entre ces variables afin de ne pas énumérer explicitement l'ensemble des états ou des couples (états, actions).

Une autre façon de réduire la taille des problèmes consiste à utiliser les Semi-MDP (SMDP) (Barto & Mahadevan, 2003) dans lesquels le nombre de pas de temps entre une décision et la suivante est une variable aléatoire. Le cadre des SMDP est la base des algorithmes d'apprentissage par renforcement hiérarchique car il permet de décomposer le MDP global en sous-problèmes qui sont plus faciles à résoudre individuellement (Barto & Mahadevan, 2003).

*Ce travail bénéficie du contrat CIFRE - 1032/2006

Dans le cadre des FMDP comme dans celui des SMDP, on fait généralement l'hypothèse que la structure du problème est connue a priori, ce qui est rarement le cas en pratique. Du côté de la structure hiérarchique, (Dietterich, 2000) considère ainsi que le principal problème de l'apprentissage par renforcement consiste à trouver une méthode générale pour découvrir automatiquement la structure hiérarchique d'un MDP. HEXQ (Hengst, 2002) et VISA (Jonsson & Barto, 2006) sont deux algorithmes destinés à résoudre ce problème. Pour les FMDP, lorsque la structure temporelle des fonctions de transition et de récompense n'est pas connue a priori, Degris et al. ont proposé SDYNA pour résoudre des problèmes d'apprentissage par renforcement de grande taille (Degris *et al.*, 2006b).

Dans cette contribution, après avoir présenté les FMDP, les SMDP et la découverte automatique d'options dans la section 2, nous décrivons dans la section 3 notre approche permettant de combiner apprentissage de la structure hiérarchique et apprentissage de la structure temporelle d'un MDP. Dans la section 4, nous comparons notre approche dans trois cas : si l'on apprend le problème sans tenir compte de sa structure hiérarchique, si la structure hiérarchique est donnée a priori et si la structure hiérarchique est apprise. Cette comparaison est réalisée sur le problème du taxi (Dietterich, 1988). Nous discutons à la section 5 de la nécessité d'introduire la gestion de plusieurs niveaux de hiérarchie pour traiter des problèmes de très grande taille et nous indiquons nos perspectives d'application à des problèmes réels de simulation d'entraînement au combat.

2 Cadre théorique

2.1 MDP et FMDP

Un MDP est un tuple $\langle S, A, R, T\rangle$, où S et A sont des ensembles finis d'états et d'actions d'un agent ; $R : S \times A \rightarrow \mathbb{R}$ est la fonction de récompense immédiate $R(s, a)$ et $T : S \times A \times S \rightarrow [0, 1]$ est la fonction décrivant les probabilités de transition $P(s\prime|s, a)$ du MDP. Réaliser l'action $a \in A$ dans l'état $s \in S$ produit la transition du système dans l'état $s\prime \in S$ avec une probabilité $P(s\prime|s, a)$ et donne lieu à la récompense $R(s, a)$. Une politique stationnaire $\pi : S \times A \rightarrow [0, 1]$ définit la probabilité $\pi(s, a)$ que l'agent fasse l'action a dans l'état s. Le but considéré ici est de trouver une politique π qui maximise la fonction de valeur $V_\pi(s)$ utilisant le critère de récompense actualisée : $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t, r|s_0 = s]$ où $\gamma \in (0, 1]$ est le facteur d'actualisation, r_t la récompense obtenue à l'instant t et s_0 l'état initial, en considérant un horizon infini. La fonction de valeur d'action $Q_\pi(s, a)$ est définie par :

$$Q_\pi(s, a) = \sum_{s\prime \in S} P(s\prime|s, a)(R(s\prime, a) + \gamma V_\pi(s\prime)) \tag{1}$$

Une politique π est optimale si, pour tout $s \in S$ et toute politique $\pi\prime$, $V_\pi > V_{\pi\prime}$. La fonction de valeur d'une politique optimale est appelée fonction de valeur optimale et notée V^*.

Un MDP factorisé (FMDP) (Boutilier *et al.*, 1995) est décrit par un ensemble de variables d'états $S = \{X_i...., X_n\}$, où chaque X_i prend ses valeurs dans un domaine fini $Dom(X_i)$. Un état $s \in S$ assigne une valeur $x_i \in Dom(X_i)$ à chaque variable d'état X_i. Le modèle de transition d'état T_a d'une action a est défini par le graphe de transition G_a representé par un réseau bayésien dynamique (DBN pour *Dynamic Bayes Net*) (Dean & Kanazawa, 1989). G_a est un graphe orienté acyclique à deux couches dont les nœuds sont $\{X_1, ..., X_n, X\prime_1, ..., X\prime_n\}$ où X_i est une variable au temps t et $X\prime_i$ la même variable au temps $t + 1$. Nous supposons qu'il n'y a pas d'arcs synchrones, c'est-à-dire pas d'arcs entre des variables au même pas de temps. Un graphe G_a est quantifié par une distribution de probabilité conditionnelle représentée par un arbre de probabilité conditionnelle (CPT) associé à chaque nœud $X\prime_i \in G_a$. Le modèle des transitions T du FMDP est alors défini par un DBN spécifique $T_a = \langle G_a, \{cpt^a_{X_1}, ..., cpt^a_{X_n}\}\rangle$ pour chaque action a. Représenter le problème sous la forme d'un FMDP permet une réduction exponentielle de la taille du modèle.

Des méthodes de planification dédiées aux FMDP ont été développées pour traiter les cas où les fonctions de transition et de récompense sont connues a priori. Nous ne mentionnerons ici que les méthodes de program-

mation dynamique, à savoir *Structured Policy Iteration* (SPI), *Structured Value Iteration* (SVI) (Boutilier *et al.*, 2000) et *Stochastic Planning Using Decision Diagrams* (SPUDD) (Hoey *et al.*, 1999), qui étendent la programmation dynamique et exploitent des fonctions de transition exprimées sous la forme d'arbres de décision ou de diagrammes de décision.

2.2 SMDP

Comme nous l'avons mentionné dans l'introduction, une autre façon de réduire la taille d'un problème est d'avoir recours à des représentations hiérarchiques. Cette méthode consiste à introduire différentes formes d'abstraction en ignorant les états ou les variables non pertinents. Pour tirer profit de ces simplifications, les méthodes d'apprentissage par renforcement hiérarchique (HRL) décomposent la tâche originale en tâches plus petites et permettent l'apprentissage et la planification à tous les niveaux d'abstraction temporelle (Barto & Mahadevan, 2003; Dietterich, 2000). En pratique, on généralise les MDP en SMDP en introduisant entre deux transitions d'état des actions de durée étendue qui correspondent à des sous-tâches.

Parmi les différents formalismes proposés pour les SMDP (Barto & Mahadevan, 2003), nous utilisons le cadre des *options* (Sutton *et al.*, 1999). Les options constituent une généralisation des actions primitives pour inclure des actions abstraites à durée étendue. Là encore, la prise en considération d'options permet de réduire exponentiellement la taille des problèmes d'apprentissage à résoudre. Même si le fait d'ajouter les options aux actions primitives augmente la taille de l'espace des choix d'actions possibles, le nombre de choix à réaliser pour atteindre l'objectif diminue, ce qui simplifie le SMDP à résoudre. Par ailleurs, des options peuvent être réutilisées d'une tâche à une autre.

Une option est un tuple $\langle \mathcal{I}, \pi, \beta \rangle$, où $\mathcal{I} \subseteq S$ est un *ensemble d'initialisation*, c'est-à-dire le sous-ensemble des états à partir desquels il est possible d'atteindre la sortie de l'option o, $\pi : S \times A \rightarrow [0,1]$ est la politique suivie lorsqu'on exécute l'option o et $\beta : S \rightarrow [0,1]$ est une fonction de probabilité d'arrêt qui représente la probabilité de stopper o dans chaque état. Une action primitive $a \in A$ du MDP est aussi une option, appelée option à un pas, avec $\mathcal{I} = \emptyset$ et $\beta(s) = 1$. Si l'option est exécutée, ses sous-options sont sélectionnées conformément à sa politique interne π jusqu'à ce qu'elle se termine dans l'état $s\prime$ avec une probabilité $\beta(s\prime)$. Quand l'option se termine, l'agent peut choisir une autre option. Le modèle SMDP est donc représenté par une hiérarchie d'options dans laquelle les options d'un niveau font appel à des options d'un niveau inférieur.

Dans cette perspective, une option o peut être vue comme une sous-tâche du SMDP $\mathcal{M}_o = \langle S_o, O_o, \Psi, R_o, T_o \rangle$ où $S_o \in S$ est l'ensemble des états de l'option, O_o est l'ensemble des sous-options de o, Ψ est l'ensemble des couples (état,option) admissibles, c'est-à-dire l'ensemble des couples dans lesquels on peut atteindre la sortie de l'option à partir de l'état considéré, déterminé par les ensembles d'initialisation des options de O_o, R_o est la fonction de récompense de l'option et T_o est la fonction de probabilité de transition.

2.3 SDYNA

Structured DYNA (SDYNA) (Degris *et al.*, 2006b) est un algorithme général d'apprentissage par renforcement indirect dans le cadre des FMDP. De même que les architectures DYNA (Sutton, 1991), SDYNA combine la planification, l'action et l'apprentissage pour résoudre par essais et erreurs des problèmes d'apprentissage par renforcement stochastiques dont les fonctions de transition et de récompense ne sont pas connues a priori. En particulier, SDYNA n'a pas besoin de construire explicitement la structure globale du DBN et apprend directement la structure locale d'un problème en utilisant un algorithme d'induction incrémentale d'arbres de décision nommé ITI (*Incremental Tree Inducer* (Utgoff, 1989)). L'algorithme construit une représentation structurée des FMDP sous la forme $\mathcal{F} = \{\forall X_i \in X : Tree[P(X_i{}'|s,a)], \forall R_i \in R : Tree[R_i]\}$ à partir d'un flux d'exemples $\langle \mathcal{A}, \sigma \rangle$ où $\mathcal{A} = (x_i, ..., x_n, a)$ est un ensemble d'attributs et σ la classe de l'exemple. Ces représentations sont exploitées simultanément avec leur construction par des versions incrémentales de techniques de planification fondées sur les FMDP. Par exemple, SPITI, une instanciation de SDYNA, utilise

une version incrémentale modifiée de SVI. Nous renvoyons à (Degris *et al.*, 2006a,b) pour une description détaillée des diverses instanciations de SDYNA.

2.4 HEXQ et VISA

Un algorithme de référence pour résoudre les SMDP est MAXQ Dietterich (2000). HEXQ (Hengst, 2002) est un algorithme d'apprentissage par renforcement hiérarchique plus récent qui décompose et résout automatiquement un FMDP sans connaître son modèle. HEXQ découvre les abstractions d'état et les options en identifiant des sous-structures qui se répètent dans l'environnement.

Pour réaliser la décomposition hiérarchique du FMDP, HEXQ détermine un ordre des variables d'état sur la base de la fréquence avec laquelle elles changent de valeur : plus elles changent, plus elles sont en fin de classement. Pour chaque variable d'état, HEXQ identifie des couples de *sortie* $\langle s_e, a \rangle$ signifiant que réaliser l'action a dans l'état s_e produit une transition imprévisible, où l'action a fait : (i) changer la valeur de la variable d'état suivante dans l'ordre déterminé ou (ii) se terminer l'option. Ces sorties constituent les frontières entre les régions qui représentent les sous-tâches du MDP global. HEXQ définit un MDP dans chaque région en associant à la sortie une transition vers un état absorbant. La solution de ce sous-MDP est une politique sur la région qui amène l'agent à la sortie à partir de n'importe quel état initial. Dans HEXQ, les politiques des sous-MDP sont apprises en ligne et deviennent les actions abstraites du niveau supérieur. Comme les actions abstraites peuvent nécessiter des durées variables, la tâche globale est alors un SMDP qui fait appel à moins de variables que le MDP original et qui n'utilise que des actions abstraites.

Variable Influence Structure Analysis (VISA) (Jonsson & Barto, 2006), décompose dynamiquement un FMDP en niveaux hiérachiques. VISA détermine des relations causales entre les variables d'états en construisant un graphe d'influence sur la base d'un modèle fourni a priori du FMDP. Pour définir des options, VISA utilise un formalisme proche de celui de HEXQ mais, au lieu de classer les variables en fonction de leur fréquence, il se fonde sur les relations du graphe d'influence. VISA utilise des algorithmes d'apprentissage par renforcement pour trouver la politique associée aux options, sans avoir recours à un modèle des probabilités de transition.

3 Méthode

Dans cette contribution, nous présentons une approche qui consiste à découper un MDP global en un ensemble d'options découvertes automatiquement et à utiliser simultanément SVI sur chaque option pour apprendre la politique locale associée. Ces options font à leur tour appel à des options plus simples qui sont les actions primitives du MDP. L'approche permet d'accélérer l'apprentissage de la politique globale, ce que nous vérifierons expérimentalement.

Outre l'apprentissage du modèle des transitions qui est réalisé incrémentalement et globalement par induction d'arbres de décision exactement comme dans SPITI, l'algorithme global de planification se décompose en deux parties : la découverte des options et de leur hiérarchie et l'exécution globale de la politique. Nous présentons successivement ces deux parties.

3.1 Découverte des options et de leur hiérarchie

Notre approche de la découverte des options est similaire à celle de HEXQ : nous définissons un ensemble de « sorties » (*exits*) correspondant à des changements de valeur pour des variables liées à la fonction de récompense, nous associons à chacune de ces sorties un contexte correspondant à l'ensemble des états dans lequel le changement de valeur correspondant se produit et nous associons une option à chaque sortie.

La découverte des changements de valeur pour les variables considérées se fait à partir des arbres représentant le modèle des transitions, comme indiqué dans l'algorithme 1.

3.1.1 Hiérarchie des sorties

Algorithme 1 : *UpdateExitsHierarchy*($\mathcal{F}_t$) $\rightarrow \mathcal{M}_e$

entrée : $\mathcal{F}_t$, le modèle des transition courant
sortie : $\mathcal{M}_e$, la hiérarchie des sorties
pour chaque $\mathcal{F}_t.Tree[P(X_i{}'|s,a)]$ **faire**
  pour chaque *feuille* $l_b \in \mathcal{F}_t.Tree[P(X_i{}'|s,a)]$ **faire**
    si x'_i *dans* $l_b \neq x^b{}_i$ *contenu dans la branche* b *(i.e. si l'action* a *modifie la valeur de* X_i*)* **alors**
      si $a \notin \mathcal{M}_e$*.node(*X_i*).ListChilds* **alors**
        $\mathcal{M}_e$*.node*(X_i)*.AddChild*(a);
      si *le nombre de variables* V *dans le contexte* c *de* $l_b > 1$ **alors**
        $v_{ch} \leftarrow \langle x^b{}_i, x'_i \rangle$;
        $\mathcal{M}_e$*.node*(a)*.AddChild*(v_{ch});
        pour chaque *variable* $v \in c$ **faire**
          si $v \neq X_i$ **alors**
            $\mathcal{M}_e$*.node*(v_{ch})*.AddChild*(v);
        Ajouter dans $\mathcal{M}_e$ l'exit $e \leftarrow \langle X_i, a, v_{ch}, c \rangle$;
      sinon
        Ajouter dans $\mathcal{M}_e$ l'exit $e \leftarrow \langle X_i, a, \emptyset, \emptyset \rangle$;

Contrairement à HEXQ, où les sorties sont des couples (état, action), nous définissons une sortie comme un tuple $\langle v, a, v_{ch}, c \rangle$, où v est la variable dont la valeur est modifiée par cette sortie, $c = \{x_1, ..., x_n\}$ est le contexte, c'est-à-dire l'ensemble des contraintes (i.e. assignations de valeurs à un sous-ensemble de variables d'états) qui conditionne cette sortie, a est l'action de sortie qui modifie la valeur de v et v_{ch} est une modification de la valeur de la variable, i.e. un couple $\langle x, x' \rangle$ où x et $x\prime$ sont les valeurs de la variable avant et après l'exécution de a respectivement[1]. On note que les actions primitives ont un contexte vide.

Cette représentation étendue des sorties rend possible l'organisation hiérarchique globale de l'ensemble des sorties $\mathcal{M}_e$ sous la forme d'un arbre de décision conformément à l'algorithme 1, qui tire parti du fait que le modèle des transitions est lui-même organisé sous la forme d'un ensemble d'arbres. La structuration des sorties en arbres est compacte et facilite la mise à jour incrémentale locale des seules parties de l'arbre qui sont concernées par un changement.

A noter que, lorsqu'on calcule le contexte d'une sortie, on exclut de ce contexte la variable dont la modification définit la sortie, ce qui a pour effet d'éviter de créer des dépendances croisées entre des options.

$\mathcal{M}_e$ est mis à jour à chaque fois que le modèle du FMDP change. Un exemple de hiérarchie des sorties dans le cas du problème du taxi est donné à la figure 2.

3.1.2 Hiérarchie des options

La hiérarchie $\mathcal{M}_o$ des options est calquée sur la hiérarchie des sorties $\mathcal{M}_e$ grâce à l'algorithme 2. La méthode $o.AddSubOptions(\mathcal{M}_e)$ ajoute à une option ses sous-options de la façon suivante : pour chaque variable du contexte de l'option, s'il y a dans $\mathcal{M}_e$ une sortie qui modifie la valeur de cette variable, on ajoute une sous-option correspondant à cette sortie.

[1] Dans le cas d'un problème stochastique, on a en fait une distribution de probabilités sur des v_{ch} pour une même sortie.

Algorithme 2 : *UpdateOptionsHierarchy*($\mathcal{F}_t, \mathcal{M}_e$) $\rightarrow \mathcal{M}_o$

entrée : $\mathcal{M}_e, \mathcal{F}_t$
sortie : $\mathcal{M}_o$
pour chaque *exit* $e \in \mathcal{M}_e$ **faire**
  Chercher l'option o contenant l'exit e;
  si $o \notin \mathcal{M}_o$ **alors**
    Initialiser une option o avec e;
    Initialiser la politique locale $o.\pi$ avec $\mathcal{F}_t$;
  else
    Mettre à jour o avec e et $\mathcal{F}_t$;
  $o.AddSubOptions(\mathcal{M}_e)$;
  ComputeInitiationSet(o);
Mettre à jour $\mathcal{M}_o$ et $\mathcal{M}_e$ en supprimant les nœuds non pertinents;

La suppression des nœuds non pertinents est rendue nécessaire par le fait que $\mathcal{M}_o$ et $\mathcal{M}_e$ évoluent en fonction de la structure du modèle des transitions qui peut être largement erronée au début de l'apprentissage, ce qui peut conduire à la construction d'options inappropriées. La pertinence des options est mesurée par deux critères :

- la mesure de nouveauté η est initialisée à 100 à la création de l'option et diminuée de 1 à chaque modification de $\mathcal{M}_o$;
- la mesure de fréquence $\phi = \frac{n}{N}$ où n est le nombre de pas où l'option est active et N le nombre de pas où son parent dans la hiérarchie est actif.

Une option est supprimée si $\eta < 0$ et $\phi < \tau$ (ici, $\tau = 0.2$).

Un exemple de hiérarchie des options dans le cas du problème du taxi est donné à la figure 3.

3.1.3 Ensemble d'initialisation et condition d'arrêt

Algorithme 3 : *ComputeInitiationSet*(o) $\rightarrow I_o$

entrée : o
sortie : I_o
$I_o \leftarrow \emptyset$;
pour chaque $v \in exit.c$ **faire**
  $I_o \leftarrow v$
pour chaque $o_i \in$ *SubOptions(o)* **faire**
  si o_i *est une action primitive* **alors**
    $I_o \leftarrow o_i.exit.v$ (avec toutes les valeurs possibles de v);
  sinon
    $I_o \leftarrow o_i.exit.v$;
    pour chaque $v_i \in o_i.exit.c$ **faire**
      $I_o \leftarrow v_i$

L'ensemble d'initialisation I_o d'une option o est construit conformément à l'algorithme 3. Cet algorithme est déclenché à chaque création ou mise à jour d'une option. L'ensemble d'initialisation I_o d'une option o contient les ensembles d'initialisation I_i de chacune de ses sous-options i. Par convention, une sous-option qui ne possède pas d'ensemble d'initialisation est déclenchable partout. C'est le cas notamment pour les options correspondant aux actions primitives. Par conséquent, toutes les valeurs des variables associées aux sorties de telles options sont acceptées. Ainsi, I_o regroupe tous les états à partir desquels l'état de sortie de l'option o est accessible.

Une option se termine en exécutant son action de sortie a dès qu'elle atteint le contexte c de sa sortie e ou

dès qu'elle ne peut plus atteindre c. On sait qu'on ne peut plus atteindre la sortie dès que l'état ne figure plus parmi l'ensemble d'initialisation de l'option. La probabilité $\beta(s)$ d'arrêt d'une option o vaut donc 1 pour les états s qui vérifient les contraintes de c et pour les états $s \notin I_o$ dans lesquels le processus ne peut plus rejoindre la sortie. Sinon, $\beta(s) = 0$.

3.2 Calcul de la politique

Algorithme 4 : TeXDYNA

initialisation : FMDP $\mathcal{F}_0$, SMDP $\mathcal{M}_0$;
initialiser $\mathcal{M}_e$ en ajoutant au nœud (**Root**) un nœud-enfant pour chaque variable d'état X_i
pour chaque *pas de temps* t **faire**
  si *il n'y a pas option en cours* **alors** $o \leftarrow$ *GetAdmissibleOption*$(\mathcal{M}_{t-1})$;
  sinon si *o n'est pas une action primitive* **alors**
    si *TerminationFunction(o)* **alors**
      $a \leftarrow o.exit.a$;
    sinon
      $a \leftarrow$ *Acting*$(o.\pi)$;
    Exécuter a ; observer $s\prime$ et r;
    $o.\pi \leftarrow$ *Planning*$(\mathcal{F}_t)$;
  sinon si *o est une action primitive* **alors**
    $a \leftarrow o.exit.a$;
    Exécuter a ; observer $s\prime$ et r;
  $\mathcal{F}_t \leftarrow$ *UpdateFMDP*$(\mathcal{F}_{t-1})$ avec (s,a,$s\prime$,r);
  option en cours $\leftarrow o$;
  $\mathcal{M}_e \leftarrow$ *UpdateExitsHierarchy*$(\mathcal{F}_t)$;
  $\mathcal{M} \leftarrow$ *UpdateOptionsHierarchy*$(\mathcal{F}_t, \mathcal{M}_e)$;

L'algorithme global, que nous nommons TeXDYNA[2], décrit dans l'algorithme 4, réalise simultanément l'apprentissage de la structure du SMDP, celle du FMDP et le calcul d'une politique associée. L'algorithme SVI qui repose sur des modèles des transitions et des récompenses appris par expérience n'est appliqué que localement au calcul des options de niveau intermédiaire dans la hiérarchie, la racine n'étant pas considérée comme une option.

La résolution de chaque sous-MDP, représenté par une option, suit les trois phases de l'algorithme SDYNA : action (méthode *Acting*()), apprentissage (méthode *UpdateFMDP*()) et planification (méthode *Planning*()). La méthode *Acting*() choisit l'action selon la politique interne de l'option $o.\pi$ en appliquant une politique d'exploration ϵ *-greedy*. Pendant la phase d'apprentissage, les représentations structurées des transitions et des récompenses sont mises à jour au niveau du MDP global par la méthode *UpdateFMDP*() qui implémente l'algorithme ITI. La phase de planification, locale à chaque option, utilise une version incrémentale modifiée de SVI qui construit $o.\pi$ sous la forme d'un arbre de décision.

De plus, pour assurer la propagation des récompenses externes aux politiques locales des options, lorsque une option de haut niveau est découverte, une récompense supplémentaire, appelée « récompense interne » r_i (par opposition aux récompenses externes reçues de l'environnement), est attribuée à son action de sortie. En pratique, on fixe $r_i = \frac{r_{max}}{2}$, où r_{max} est la plus grande récompense immédiate externe possible. Cette heuristique s'inspire de l'idée de « *salient event* » proposée dans (Singh *et al.*, 2005).

Enfin, la méthode *GetAdmissibleOption*() retourne les options o dont l'ensemble d'initialisation I_o contient l'état courant s. En pratique, les options de haut niveau forment une partition de l'espace d'état, si bien que *GetAdmissibleOption*() renvoie une seule option et qu'il n'est pas nécessaire d'apprendre quelle option déclencher dans quel état.

[2] pour *Temporally Extended* SDYNA

4 Expérimentations

4.1 Le problème du taxi

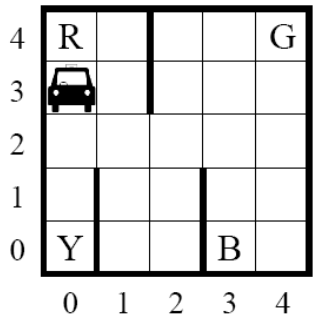


FIG. 1 – Problème du taxi

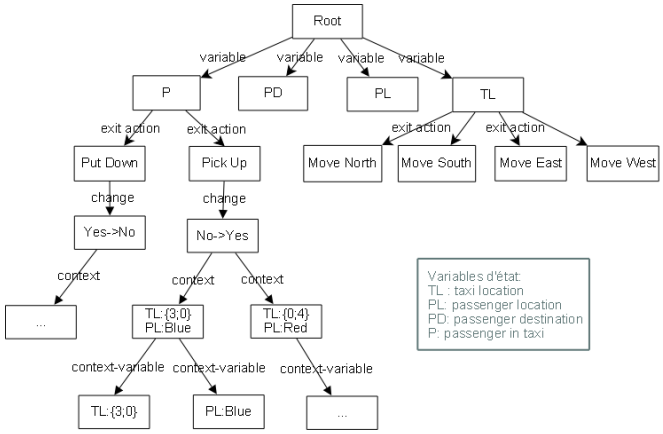


FIG. 2 – Sorties associées au problème du taxi

Le problème du taxi, représenté sur la figure 1, a été proposé par (Dietterich, 1988). La représentation du problème est la suivante : un taxi se déplace sur une grille 5×5 qui comporte quatre zones spécifiques, notées R, G, Y et B, définissant la localisation et/ou la destination d'un passager. Le problème du taxi se décompose en épisodes. A chaque nouvel épisode, le taxi se trouve à une position choisie au hasard, un passager se trouve dans l'une des quatre zones spécifiques et veut être transporté dans une autre zone choisie également au hasard. Un épisode se termine quand le passager est déposé à la destination ou lorsque le nombre de pas de temps prédéfini a été dépassé. A chaque pas de temps, le taxi peut exécuter l'une des 6 actions primitives : aller au nord, au sud, à l'est ou à l'ouest, prendre le passager ou déposer le passager. Dans la version stochastique, par opposition à la version déterministe, il y a une probabilité de 0.2 à chaque pas de temps pour que le taxi exécute une action aléatoire plutôt que celle qu'il a choisie. Le taxi reçoit une récompense égale à 20 s'il dépose le passager à l'endroit correct. Si le taxi exécute l'action de prendre ou déposer le passager dans une zone incorrecte, il reçoit une punition de -10. Une punition de -1 est attribuée pour chaque action primitive. La récompense interne dans le cas d'apprentissage avec options vaut 10. Enfin, il y a 500 états possibles.

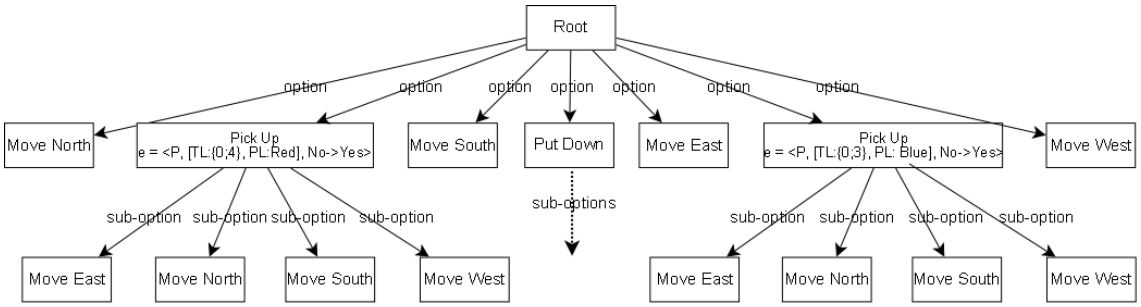


FIG. 3 – Options associées au problème du taxi

4.2 Résultats

Les figures 4 et 5 sont obtenues en faisant une moyenne sur 20 essais sur le problème du taxi dans ses versions déterministe et stochastique respectivement. Pour lisser le résultat, on effectue une moyenne glissante, les valeurs sont pondérées par les dix valeurs voisines. On donne les résultats de SVI sans options (*action*

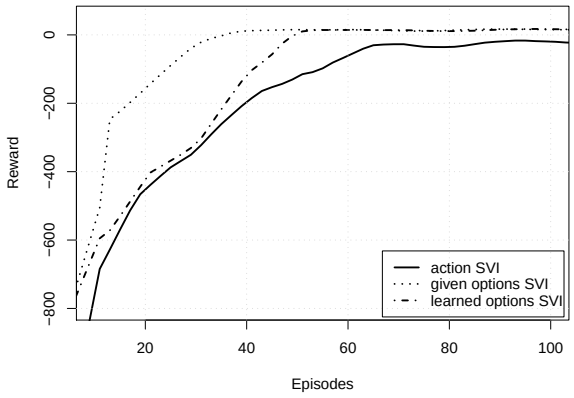


FIG. 4 – Problème de taxi déterministe

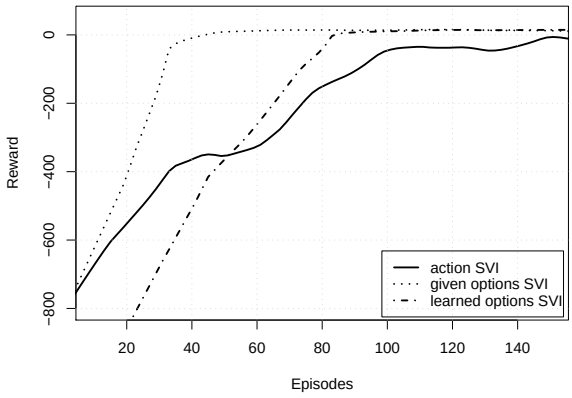


FIG. 5 – Problème de taxi stochastique

SVI), avec des options données a priori (*given options* SVI) et avec les options apprises par notre méthode (*learned options* SVI). Le nombre d'actions pour chaque épisode est limité à 300.

On constate que notre modèle a besoin d'environ 50 épisodes pour converger dans le cas déterministe et 80 épisodes dans le cas stochastique. Comme attendu, cette performance est intermédiaire dans les deux cas entre la performance d'un système auquel on donne les options a priori et SVI qui fonctionne sans option. Par ailleurs, dans le cas stochastique, notre modèle se compare très favorablement à MAXQ, qui a besoin de 115 épisodes environ pour converger et à HEXQ qui a besoin de 160 épisodes environ. On constate en outre que MAXQ et HEXQ convergent tous les deux moins vite que SVI seul, ce qui s'explique pour le premier par le fait qu'il n'utilise pas explicitement la structure factorisée du problème et pour le second par le fait qu'il perd un grand nombre d'épisodes à estimer correctement l'ordre des variables à partir desquelles il construit sa hiérarchie.

5 Discussion et conclusion

Nos travaux de recherche doivent à terme être appliqués à la génération automatique de comportements pour des entités animées (civils, guerilleros, militaires, voire véhicules) d'un logiciel de simulation du théâtre opérationnel.

Sur le problème du taxi, nous avons montré que notre approche qui combine factorisation et abstraction se comporte mieux que des algorithmes de référence de ces domaines. Pour ce qui est de la factorisation, nous bénéficions de la puissance de la méthode employée dans SPITI qui consiste à apprendre incrémentalement le modèle des fonctions de transition et de récompense sous la forme d'arbres de décision, fonctionnant ainsi sans connaissance a priori sur la structure du problème. Pour ce qui est de l'abstraction, en revanche, notre méthode peut être largement améliorée. En effet, en l'état, notre algorithme ne construit qu'une hiérarchie à deux niveaux, le premier niveau étant constitué des actions primitives et le second d'un ensemble d'options mutuellement exclusives. Si cette méthode suffit à traiter efficacement le problème du taxi, elle nous semble insuffisante pour aborder les problèmes de beaucoup plus grande taille qui constituent nos objectifs applicatifs. Face à de tels problèmes, nous aimerions disposer d'une hiérarchie d'options dont le nombre de niveaux ne soit pas contraint a priori et dont l'option racine soit une option comme les autres. On apprendrait ainsi la politique de l'option racine de la même façon que pour les autres options. Par ailleurs, notre méthode de suppression des options inadaptées est provisoire, elle fait appel à deux paramètres qu'il faut régler empiriquement. Il sera nécessaire de proposer une méthode mieux fondée pour traiter ce point.

Au delà de ces améliorations locales de notre algorithme, nos perspectives concernent la prise en compte

du caractère non markovien des simulations complexes que nous allons aborder et la mise au point d'une méthodologie pour introduire la connaissance de l'utilisateur là où c'est nécessaire.

Références

BARTO A. & MAHADEVAN S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Systems Journal*, **13**, 41–77. Special Issue on Reinforcement Learning.

BOUTILIER C., DEARDEN R. & GOLDSZMIDT M. (1995). Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, p. 1104–1111.

BOUTILIER C., DEARDEN R. & GOLDSZMIDT M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, **121**(1-2), 49–10.

DEAN T. & KANAZAWA K. (1989). A model for reasoning about persistence and causation. *Computational Inteligence*, **5**, 142–150.

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 Uncertainty in Artificial Intelligence*, p. 122–129, Massachusetts Institute of Technology, Cambridge : 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, Pittsburgh, Pennsylvania : ACM.

DIETTERICH T. (1988). The MAXQ method for hierarchical reinforcement learning. In M. KAUFMANN, Ed., *In 15th International Conference on Machine Learning*.

DIETTERICH T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, **13**, 227–303.

HENGST B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the 19th International Conference on Machine Learning*, p. 243–250.

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, Stockholm.

JONSSON A. & BARTO A. (2006). Causal graph based decomposition of factored MDPs. *Journal of Machine Learning Research*, **7**, 2259–2301.

SINGH S., BARTO A. & CHENTANEZ N. (2005). Intrinsically motivated reinforcement learning. *Advances in Neural Information Processing Systems*, **18**, 1281–1288.

SUTTON R., PRECUP D. & SINGH S. (1999). Between MDPs and semi-MDPs : A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, **112**, 181–211.

SUTTON R. S. (1991). DYNA, an integrated architecture for learning, planning and reacting. In *Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures*.

SUTTON R. S. & BARTO A. G. (1998). *Reinforcement Learning : An Introduction*. Cambridge, MA : MIT Press.

UTGOFF P. E. (1989). Incremental induction of decision trees. *Machine Learning*, **4**, 161–186.

Un Algorithme Amélioré d'Itération de la Politique Approchée pour les Processus Décisionnels Semi-Markoviens Généralisés

Emmanuel Rachelson[1], Patrick Fabiani[1], and Frédérick Garcia[2]

[1] ONERA-DCSD
2, avenue Edouard Belin, F-31055 Toulouse, FRANCE
emmanuel.rachelson, patrick.fabiani@onera.fr,

[2] INRA-BIA
Chemin de Borde Rouge, F-31326 Castanet-Tolosan, FRANCE
fgarcia@toulouse.inra.fr

Résumé : La complexité des problèmes de décision dans l'incertain dépendant du temps provient souvent de l'interaction de plusieurs processus concurrents. Les Processus Décisionnels Semi-Markoviens Généralisés (GSMDP) consituent un formalisme efficace et élégant pour représenter à la fois les aspects de concurrence d'événements et d'actions et d'incertitude. Nous proposons un formalisme GSMDP étendu à un temps observable et un espace d'états hybride. Sur cette base, nous introduisons un nouvel algorithme inspiré de l'itération de la politique approchée afin de construire des politiques efficaces. Cet algorithme repose sur une exploration guidée par la simulation et utilise les techniques d'apprentissage à vecteurs supports. Nous illustrons cet algorithme sur un exemple et en proposons une version améliorée qui compense sa principale faiblesse.

1 Introduction

Les Processus Décisionnels Semi-Markoviens Généralisés (GSMDP) Younes & Simmons (2004) constituent un formalisme élégant pour décrire les problèmes de planification présentant les trois aspects d'incertitude, de temps continu et d'activités concurrentes. Imaginons, par exemple, devoir construire la stratégie d'exploitation d'un métro où les seules actions disponibles consistent à introduire ou retirer des rames de la circulation. L'objectif est de maximiser le nombre de passagers transportés tout en minimant le coût de fonctionnement. Les dates d'arrivée des passages, leurs montées et descentes et les retards des trains rendent stochastique le résultat de chaque action et de chaque événement en terme d'état final et de future date de décision. Par ailleurs, le flux des passagers et leurs destinations dépendent fortement de l'heure de la journée. D'autres exemples combinent les difficultés de la planification dans l'incertain, des contraintes temporelles fortes et de la concurrence des processus : la gestion du trafic au sol des avions dans un aéroport, la coordination d'un drone aérien avec la dunamique de son environnement résultant de l'interaction concurrente de processus exogènes (d'autres agents, l'évolution de l'environnement, de la mission, . . .).

Nous commençons par présenter le formalisme GSMDP en section 2 et l'augmentons d'une variable temporelle continue afin de prendre en compte la dépendance explicite au temps. Ce formalisme est mis en lumière des approches MDP classiques, des techniques pour gérer les varaibles continues et des représentation des processus stochastiques concurrents. Puis (section 3 nous présentons un algorithme d'apprentissage par renforcement reposant sur la simulation développé pour exploiter la structure et les difficultés inhérentes aux GSMDP. La section 4 évalue cet algorithme en mettant en avant sa principale faiblesse. Nous en introduisons une version améliorée en section 5.

2 Processus Décisionnels Semi-Markoviens Généralisés

2.1 Temps, concurrence et incertitude

Les Processus Décisionnels de Markov (MDP) sont un modèle couramment utilisé pour représenter les problèmes de planification dans l'incertain. Un MDP se décompense en un quadruplet $< S, A, P, r >$, où S et A sont des espaces d'états et d'actions dénombrables, $P(s'|s,a)$ est une fonction de transition probabiliste entre états et $r(s,a)$ est une fonction de récompense pour la transition (s,a) qui permet de construire des critères et d'évaluer actions et politiques. Les solutions aux problèmes MDP sont généralement donnés sous la forme de *politiques Markoviennes* associant une action à entreprendre à tout état de S. On peut introduire des critères d'évaluation de ces politiques, comme le critère γ-pondéré ($V_\gamma^\pi(s) = E\left(\sum_{\delta=0}^{\infty} \gamma^\delta r(s_\delta, \pi(s_\delta))\right)$). Ces critères permettent la définition de la *fonction de valeur* V^π associée à une politique. Un résultat fondamental indique que pour toute politique dépendant de l'historique des états et actions, il existe une politique Markovienne au moins aussi bonne vis-à-vis d'un critère. Par conséquent, l'on peut chercher sans risque une solution optimale aux problèmes MDP dans l'espace restreint des politiques Markoviennes sans craindre de perdre en optimalité. Enfin, les algorithmes classiques comme l'*itération de la valeur* ou l'*itération de la politique* reposent sur le fait que la fonction de valeur V^* de la politique optimale obéit à l'équation d'optimalité de Bellman (équation 1) Bellman (1957).

$$V^*(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right] \quad (1)$$

Parmi les différentes approches de prise en compte du temps dans le cadre MDP on peut citer les Semi-MDP (Puterman (1994)), les Time-dependent MDP (Boyan & Littman (2001)) et plus récemment les XMDP (Rachelson *et al.* (2008b)). Par ailleurs, le traitement d'un temps continu peut s'inspirer des algorithmes conçus pour les problèmes à espaces d'états continus ou hybrides (eg. Hauskrecht & Kveton (2006), Feng *et al.* (2004)). Une discussion plus détaillée de la relation entre temps et MDP est présentée dans Rachelson *et al.* (2008a). Ces modèles reposent sur l'introduction d'une variable temporelle continue dans le model afin de représenter l'instationnarité dans la fonction de transition. Cependant, la tâche d'écriture des fonctions de transition et de durée pour de tels modèles demande un gros travail d'ingénierie. Par exemple, l'effet de l'action $RemoveTrain$ sur l'état global du problème du métro est le résultat de plusieurs processus concurrents : les arrivées de passagers, le passage des métro, leur mise en service ou leur retrait, etc. ; tous ces processus sont en compétition pour changer l'état du système et synthétiser leurs influences stochastiques concurrentes dans les fonctions de transition et de durée est un travail complexe.

La littérature des processus stochastiques fournit l'outil des Processus Semi-Markoviens Généralisés (GSMP) (Glynn (1989)) afin de représenter le déroulement concurrent de plusieurs processus stochastiques semi-Markoviens concurrents. Younes & Simmons (2004) a introduit les Processus Décisionnels Semi-Markoviens Généralisés (GSMDP) afin de représenter le problème de décision dans l'incertain où les actions sont concurrentes d'événements stochastiques et non-controlables. Un GSMDP décompose le problème en factorisant la fonction de transition globale par les différentes contributions des événements séparés. Cela fait des GSMDP un formalisme élégant et efficace afin de décrire la complexité des problèmes stochastiques dépendant du temps. Nous présentons une définition formelle des GSMDP au paragraphe suivant et nous concentrons sur les GSMDP à temps continu et observable dans le reste de l'article.

2.2 GSMDP

Partons du point de vue des processus stochastiques sans couche décisionnelle. Formellement, un GSMP Glynn (1989) est décrit par un ensemble d'états S et un ensemble d'événements E. A tout instant, le processus est dans un état s et il existe un sous-ensemble E_s d'événements qui sont dits *actifs*. Ces événements représentent les différents processus concurrents qui sont en compétition pour déclencher la prochaine transition. Comme dans le cadre des automates temporisés, à chaque événement actif e, on associe une horloge c_e représentant la durée avant que cet événement ne déclenche une transition. Cette durée serait le temps de séjour dans l'état s si e était l'unique événement actif. L'événement e^* possédant la plus petite horloge c_{e^*} (le premier à déclencher) emmène le processus global dans un nouvel état. La transition est alors décrite par le modèle de transition associé à l'événement en question : l'état suivant est tiré selon la distribution de probabilité $P_{e^*}(s'|s)$. Dans le nouvel état s', les événements n'appartenant pas à $E_{s'}$ sont désactivés (leurs horloges sont mises à $+\infty$) et les horloges des événements de E_{e^*} sont mises à jour de la façon suivante :

– Si $e \in E_s \setminus \{e^*\}$, alors $c_e \leftarrow c_e - c_{e^*}$
– Si $e \notin E_s$ or if $e = e^*$, tirer c_e selon $F_e(\tau|s')$

Le premier événement à déclencher emmène alors le processus dans un nouvel état où ces opérations sont reprises. Il est important de noter que le processus global d'évolution de l'état d'un GSMP n'est pas Markovien : connaître l'état courant n'est pas suffisant pour décrire la distribution de probabilité sur l'état suivant. Nielsen (1998) a montré qu'en introduisant les horloges dans l'espace d'état on récupérait la propriété de Markov pour les GSMP. Nous discutons de ce point au paragraphe suivant. Introduire un choix d'actions dans un GSMP permet de définir un GSMDP (Younes & Simmons (2004)). Dans un GSMDP, on identifie un sous-ensemble A d'événements *controlables* (les actions), les autres étant appelés incontrolables ou exogènes. Les actions peuvent être activées ou désactivées à discrétion et l'ensemble $A_s = A \cap E_s$ n'est jamais vide car il contient toujours au moins l'action "idle" a_∞ (dont l'horloge vaut en permanence $+\infty$) qui laisse le premier événement exogène emmener le processus dans un nouvel état. Tout comme dans le cas des MDP, chercher une stratégie de contrôle implique de définir des récompenses $r(s, e)$ ou $r(s, e, s')$ et d'introduire des politiques et critères.

2.3 Controler un GSMDP

Comme indiqué précédemment, la fonction de transition du processus global ne conserve pas la propriété de Markov à moins d'augmenter l'espace d'états. Dans le cadre MDP classique, cette propriété de Markov nous permet de montrer qu'il existe toujours une politique Markovienne (ne dépendant que de l'état courant) au moins aussi efficace que n'importe quelle politique histoire-dépendante (Puterman (1994)). En revanche, dans le cas des GSMDP, cela n'est plus possible et afin de définir des critères et de trouver des politiques optimales, nous avons en général besoin de permettre à la politique de dépendre de la totalité du *chemin d'exécution* du processus. Un chemin d'exécution Younes & Simmons (2004) de longueur n partant de l'état s_0 et allant dans l'état s_n est une séquence $\sigma = (s_0, t_0, e_0, s_1, \ldots, s_{n-1}, t_{n-1}, e_{n-1}, s_n)$ où t_i désigne le temps de séjour dans l'état s_i avant qu'e_i ne se déclenche. Comme dans Younes & Simmons (2004), nous définissons la valeur γ-pondérée de σ par :

$$V_\gamma^\pi(\sigma) = \sum_{i=0}^{n-1} \gamma^{T_i} \left(\gamma^{t_i} k(s_i, e_i, s_{i+1}) + \int_0^{t_i} \gamma^t c(s_i, e_i) dt \right) \tag{2}$$

où k et c sont les récompense instantanée et taux de récompenses classiques aux $SMDP$, et où $T_i = \sum_{j=0}^{i-1} t_j$. On peut alors définir la valeur espérée de la politique π dans l'état s comme l'espérance sur tous les chemins d'exécution possibles partant de s : $V_\gamma^\pi(s) = E_s^\pi \left[V_\gamma^\pi(\sigma) \right]$.

Cela permet de définir un critère d'évaluation des politiques. L'objectif est alors de trouver des politiques maximisant ce critère. Le principal problème réside dans le fait qu'il est complexe d'effectuer une recherche dans l'espace des politiques histoire-dépendantes. Par ailleurs, la méthode des variables supplémentaires est couramment utilisée afin de transformer des processus non-Markoviens en processus Markoviens. Elle consiste à enrichir l'espace d'état avec exactement assez de variables afin de pouvoir écrire la distribution sur l'état futur à partir de l'état courant. Dans Nielsen (1998), Nilsen augemente l'espace d'états avec les horloges et montre que cette opération suffit à rendre le processus Markovien. Nous noterons cet espace d'état (s, c) par commodité. Il est toutefois irréaliste de définir des politiques sur cet espace augmenté car les horloges contiennent intrinsèquement de l'information concernant le *futur* du processus et sont donc, par nature, inobservables. A partir de cette constatation, plusieurs options sont disponibles :

– Abandonner l'optimalité et de cherche des "bonnes" politiques au sein d'un ensemble restreint de politiques, *eg.* celles définies sur l'état naturel s uniquement.
– Chercher des hypothèses de représentation qui simplifient le GSMDP, rendent l'état naturel s Markovien de nouveau et se ramènent à un MDP.
– Construire des politiques optimales sur l'espace augmenté (s, c) puis inférer une politique sur les variables observables uniquement.
– Chercher un ensemble de variables *observables* permettant de conserver la propriété de Markov, *eg.* l'état naturel s, les durées d'activation τ_i et les états d'activation s_i de chaque e_i (état noté(s, τ, s_a)).

Younes & Simmons (2004) utilise la seconde option ci-dessus. Dans les paragraphes suivants, nous introduisons un nouvel algorithme d'apprentissage de politique fondé sur l'itération de la politique approchée guidée par la simulation, conçu pour gérer les grands espaces d'états des GSMDP à temps continu et observable et adapté aux trois autres options. Une discussion plus détaillée concernant les motivations de cette approche est présentée dans Rachelson *et al.* (2008a).

3 Itération de la politique approchée guidée par la simulation pour les GSMDP

3.1 Itération de la politique approchée

Notre algorithme s'inscrit dans la famille des méthodes d'Itération de la Politique Approchée (API). L'itétation de la politique est un algorithme de résolution des MDP qui explore l'espace des politiques de manière incrémentale en deux temps comme illustré sur la figure 3. Etant donné une politique π_n à l'étape n, la première étape consiste à calculer la valeur de π_n. La seconde étape effectue alors une itération de Bellman dans tous les états, améliorant ainsi la politique. Une propriété importante de l'itération de la politique est son bon comportement "anytime" : à chaque pas n de l'algorithme, la politique π_n est la meilleure politique considérée jusque là. L'itération de la politique converge généralement en moins d'itérations que l'itération de la valeur mais prend plus de temps car la phase d'évaluation est coûteuse en temps de calcul. Afin de traiter des cas réels, on autorise généralement l'évaluation de la politique à être approchée (comme dans Lagoudakis & Parr (2003)). Il y a peu de garanties théoriques sur la convergence et l'optimalité d'API, comme indiqué dans Munos (2003). L'idée de base de notre algorithme est de démarrer avec une politique définie sur tout l'espace d'états et de l'améliorer incrémentalement sur les états *pertinents* pour le problème courant.

$$\pi_{n+1}(s) = \underset{a \in A}{argmax} \left[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_n}(s') \right] \tag{3}$$

3.2 L'algorithme

3.2.1 Exploration partielle de l'espace d'états.

Nous nous intéressons à des espaces d'états continus - éventuellement hybrides ; à moins de faire des hypothèses fortes sur la forme des fonctions de transition et de récompense (comme dans Boyan & Littman (2001) ou Younes & Simmons (2004)), il n'est pas possible de calculer exactement la valeur de π_n. A la place, nous échantillonnons l'espace d'états afin d'obtenir une évaluation de V^{pi_n}. Nous simulons N_0 fois la la politique π_n depuis l'état courant du processus et stockons les triplets état-date-récompense $(s_\delta, t_\delta, r_\delta)$. De cette manière, en effectuant la somme cumulée des récompenses depuis un état à la fin de la simulation, un chemin d'exécution fournit une fonction de valeur V_i sur les états traversés (equation 2). L'ensemble de ces fonctions de valeur issues de la simulation forme une base d'apprentissage $\{(s,v)\},\ s \in S, v \in \mathbb{R}$ à partir de laquelle on souhaite construire une fonction de valeur généralisée sur tous les états. Il est important de noter que le *principe de causalité* et l'observabilité du temps impliquent qu'il ne peut y avoir de boucles qu'à temps fixé (boucles instantanées) et que la probabilité d'une infinité de boucles instantanées est nulle. Par conséquent, les simulations s'arrêtent nécessairement en atteignant les états absorbants à $t = horizon$. Lorsque $N_0 \rightarrow \infty$, le sous-espace d'états exploré par les simulations tend vers le sous-espace d'états atteignable depuis s_0 et chaque état est parcouru un nombre de fois correspondant à la probabilité de visite de la chaîne de Markov stationnaire définie par les variables (s_0, c_0, π_n) Nielsen (1998). Par conséquent :

$$\forall s \in Reachable(s_0),\ \lim_{N_0 \rightarrow \infty} \frac{\sum_{i=1}^{N_0} V_i(s)}{N_0} = V^{\pi_n}(s) \tag{4}$$

Nous résumons cette approche en disant que nous laissons la politique courante guider l'échantillonage dans l'espace d'états afin d'atteindre une bonne estimation de la fonction de valeur sur les états atteignables.

3.2.2 Fonction de valeur généralisée.

Il n'est pas possible d'effectuer un nombre infini de simulations et notre base d'apprentissage a nécessairement une taille finie. Nous souhaitons donc généraliser cette information afin d'extrapoler la fonction de valeur sur tous les états. La base d'apprentissage sert alors à entraîner un régresseur qui généralise à l'ensemble de l'espace d'états. Plusieurs approches couplant apprentissage par renforcement et régression ont été développées récemment - certaines utilisant des arbres Ernst *et al.* (2005), des algorithmes évolutionnaires Whiteson & Stone (2006), des méthodes à noyaux Ormoneit & Sen (2002), etc. - mais peu ont

été couplées avec une approche de simulation de la politique. Nous nous sommes concentrés sur la régression à vecteurs supports (SVR) à cause de sa capacité à traiter les espaces de grande dimension auxquels nous sommes confrontés. Les SVR appartiennent à la famille des méthodes à noyaux. L'entraînement d'une SVR correspond à la recherche d'un hyperplan interpolateur dans un espace de dimension plus élevée que l'espace des échantillons. En pratique, les SVR profitent du *kernel trick* afin d'éviter d'exprimer explicitement la projection dans cet espace de grande dimension. Pour plus de détails sur les machines à vecteurs supports, nous renvoyons le lecteur à Vapnik *et al.* (1996). Dans notre cas, nous noterons $\tilde{V}_n(s)$ la fonction de valeur apprise pour la politique π_n.

3.2.3 Instanciation en ligne de la politique.

L'exploration guidée par la simulation et la généralisation sous forme de SVR sont des méthodes destinées à améliorer la phase d'évaluation de l'API. La dernière spécificité de notre algorithme concerne la phase d'optimisation. Pour des espaces d'états de grande taille ou continus, il n'est pas possible d'améliorer la politique partout. De fait, en général le calcul d'une politique complète n'est pas pertinent puisque la majorité des actions calculées ne sera pas utilisée pendant la simulation. Cette idée est à la base de l'itération de la politique *asynchrone*. En revanche, il peut être plus aisé de calculer en ligne la meilleure action à entreprendre dans chaque état visité (par rapport à la fonction de valeur construite à l'itération précédente). Dans un MDP classique, la phase d'optimisation consiste à résoudre l'équation 5 dans chaque état :

$$\pi_{n+1}(s) \leftarrow \arg\max_{a \in A} \tilde{Q}_{n+1}(s,a) \tag{5}$$

$$\text{avec : } \tilde{Q}_{n+1}(s,a) = r(s,a) + \sum_{s' \in S} P(s'|s,a)\tilde{V}_n(s,a)$$

Pour des espaces continus, calculer π_{n+1} nécessite alors de calculer des intégrales sur P et $\tilde{V}_n$. Nous ne souhaitons pas faire d'hypothèses sur la forme de ces fonctions et procéderons de nouveau par échantillonage pour l'évaluation de ces intégrales. Enfin, comme la fonction P elle-même n'est pas forcément connue du décideur et comme nous supposons disposer d'un simulateur du système, nous utiliserons ce simulateur afin d'effectuer cet échantillonage et de construire la fonction $\tilde{Q}_{n+1}(s,a)$ associée à l'exécution de l'action a dans l'état s sachant que l'on obtient $\tilde{V}_n$ après un coup (equation 6). A la fin de la phase d'évaluation, la fonction de valeur généralisée $\tilde{V}_n$ est stockée et la politique n'est pas optimisée. A la place, on réentre dans une phase de simulation où, à chaque état traversé, on calcule la meilleure action à partir des Q-valeurs, ce qui correspond à appliquer la politique π_{n+1} en ne la calculant que dans les états nécessaires. Le gain en terme d'exécution de l'algorithme est facilement illustrable dans le cas des espaces d'états discrets puisqu'alors on remplace $|S|$ evaluations des Q-valeurs pour la mise à jour de la politique par le nombre d'états parcourus pendant la simulation. C'est particulièrement utile puisqu'avec un temps observable, un état n'est jamais traversé deux fois (sauf boucles instantanées). Par conséquent, le calcul de $\tilde{Q}_{n+1}(s,a)$ se fait en simulant N_a fois l'application de l'action a en s et en observant les jeux de $\{(r_i, s'_i)\}$ comme dans l'équation 6. La politique renvoie alors l'action correspondant à la plus grande Q-valeur. Nous désignons cette instanciation en ligne de la politique "itération de la politique approchée en ligne".

$$\tilde{Q}_{n+1}(s,a) = \frac{1}{N_a} \sum_{i=1}^{N_a} \left[r_i + \tilde{V}_n(s'_i) \right] \tag{6}$$

Pour résumer, l'amélioration de la politique est effectuée en ligne ; en chaque état visité nous effectuons une itération de Bellman en utilisant la fonction de valeur généralisée issue de l'étape précédente puis appliquons l'action afin d'aller dans l'état suivant. Cette approche consiste à laisser la politique en cours de construction guider le choix du sous-ensemble d'états sur lequel on améliore les politiques successives.

3.2.4 Résumé de la méthode et implantation

L'algorithme *online Approximate Temporal Policy Iteration* (online-ATPI) est présenté à l'algorithme 1. On peut noter que s y désigne l'espace d'état que l'on considère observable pour la politique. Cela fait d'online-ATPI un algorithme adaptable selon le jeu de variables d'état que l'on décide de prendre (comme en section 2.3). Nous avons implanté une version d'online-ATPI sur l'état naturel du processus. Nos travaux futurs porteront notamment sur l'extension aux états (s,c) et (s,τ,s_a). Dans la version courante

de l'algorithme, la fonction de valeur généralisée est utilisable telle qu'elle alors que dans le cas de l'état (s, c) une étape supplémentaire est nécessaire afin de construire un estimateur des horloges.

Algorithm 1 Online-ATPI

main :
Input : π_0 or $\tilde{V}_0$, s_0
loop
  $TrainingSet \leftarrow \emptyset$
  for $i = 1$ to N_0 **do**
    $\{(s, v)\} \leftarrow$ simulate$(\tilde{V}, s_0)$
    $TrainingSet \leftarrow TrainingSet \cup \{(s, v)\}$
  end for
  $\tilde{V} \leftarrow$ TrainApproximator$(TrainingSet)$
end loop

simulate$(\tilde{V}, s_0)$ **:**
$ExecutionPath \leftarrow \emptyset$
$s \leftarrow s_0$
while horizon not reached **do**
  $action \leftarrow$ ComputePolicy$(s, \tilde{V})$
  $(s', r) \leftarrow$ GSMDPstep$(s, action)$
  $ExecutionPath \leftarrow ExecutionPath \cup (s', r)$
end while
convert execution path to value function $\{(s, v)\}$ (eqn 2)
return $\{(s, v)\}$

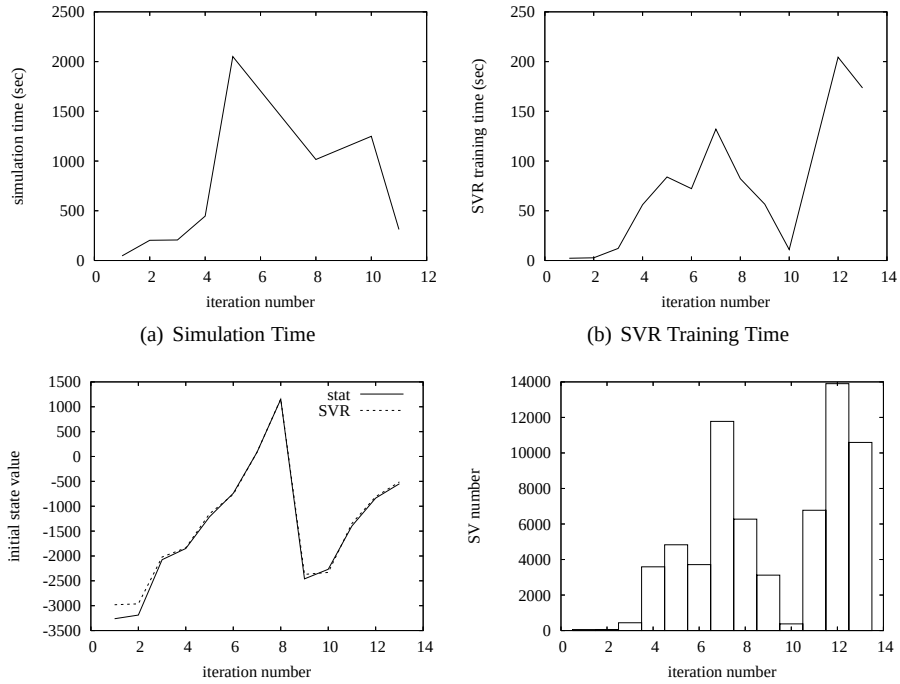
ComputePolicy$(s, \tilde{V})$ **:**
for $a \in A$ **do**
  $\tilde{Q}(s, a) = 0$
  for $j = 1$ to N_a **do**
    $(s', r) \leftarrow$ GSMDPstep(s, a)
    $\tilde{Q}(s, a) \leftarrow \tilde{Q}(s, a) + r + \gamma^{t'-t}\tilde{V}(s')$
  end for
  $\tilde{Q}(s, a) \leftarrow \frac{1}{N_a}\tilde{Q}(s, a)$
end for
$action \leftarrow \arg\max_{a \in A} \tilde{Q}(s, a)$
return $action$

4 Evaluation

4.1 Le problème de contrôle du métro

Nous avons testé online-ATPI sur un exemple simple du problème de contrôle d'un métro ayant 4 rames et 6 stations. Dans ce problème, le but est d'optimiser le coût d'exploitation du réseau sur une journée. Les événements comme $trainMovement$ et les actions comme $addTrain$ coûtent de l'argent (pour la consommation d'énergie) tandis que chaque passager quittant le métro rapporte le prix d'un ticket. La formalisation du problème fait intervenir les événements concurrents :

- $passengerArrival_i$: arrivée d'un passager en station i - exogène.
- $trainMovement_j$: mouvement du train numéro j, actif uniquement si le train est en service. Cet événement inclut également la montée et la descente des passagers lors de l'arrivée dans la nouvelle station - exogène.
- $addTrain_j$: met le train j en service à la station 1 - contrôlable.
- $removeTrain_j$: retire la rame j de la ligne, actif uniquement si la rame j est à la station 0 - contrôlable.
- a_∞ : NoOp - contrôlable.



(a) Simulation Time (b) SVR Training Time

(c) Policy quality (d) Number of suport vectors

L'espace d'état de ce problème inclut un temps continu, des positions discrètes pour les trains et des variables booléennes redondantes (eg. $NoTrainAtStation_j$). C problème a au final 19 événements concurrents et un espace d'états hybride de dimension 22. En fonction de l'heure de la journée, les fonctions de transition de chaque événement décrivent l'évolution continue des flux de passagers, de leurs destinations et de leurs fréquences d'arrivée.

Cette série d'expériences utilise le moteur de simulation multi-modèles VLE Quesnel *et al.* (2007), s'appuyant sur les principes du formalisme DEVS Zeigler *et al.* (2000). Nous avons développé des extensions pour les GSMP et GSMDP à la plate-forme VLE et les avons utilisées pour les tâches de simulation. Cela a permis de tirer parti de de l'interopérabilité des modèles DEVS. Plus précisément, cela permet la simulation de modèles hétérogènes couplés, *ie.* si un événement ou un système externe interagit avec la simulation de notre GSMDP mais n'est pas écrit comme un GSMP lui-même, alors le couplage DEVS permet la simulation *sure* du système, garantissant les propriétés des simulations de systèmes à événements discrets. Cet aspect n'a pas d'impact direct sur l'algorithme mais étend son champ d'application aux systèmes à événements discrets en général. Nous avons utilisé des valeurs de $N_0 = 20$ simulations par itération de la politique en prenant $N_a = 15$ échantillons par action (Cf. algorithme 1). La SVR est une ϵ-SVR standard, adaptée de la bibliothèque C++ LIBSVM Chang & Lin (2001), utilisant des noyaux Gaussiens avec $\sigma^2 = 20$. A l'issue des N_0 simulations, la base d'apprentissage contenait environ 45000 points. Ces tests ont été effectués sur un processeur simple coeur cadencé à 1,7GHz avec 884Mo de RAM.

4.2 Discussion

Les figures 1(a) à 1(d) présentent les résultats de l'algorithme lorsque ce dernier est initialisé avec une politique par défaut qui fait tourner tous les trains en permanence. Les figures 1(a) et 1(d) illustrent le fait que le nombre de vecteurs supports dans la SVR est le facteur handicapant pour la vitesse de simulation. Cela pourrait être amélioré au prix d'un peu de perte en précision en utilisant d'autres techniques comme les ν-SVR. Toutefois, puisque les N_0 simulations génèrent toujours environ le même nombre d'échantillons d'apprentissage, le nombre de vecteurs supports est borné.

En premier lieu, entre les itérations 1 et 8, la valeur espérée de l'état initial augmente, comme on pouvait l'espérer d'un algorithme type itération de la politique. Cette croissance n'est pas linéaire et dépend de la structure du problème. Si la politique courante emmène la simulation vers des états qui sont "loin" des états déjà explorés (états pour lesquels $\tilde{V}$ peut être une estimation erronée) et qui génèrent des récompenses

très mauvaises, il peut arriver que la valeur de l'état inital chute pour une itération. C'est le contrecoup de l'exploration partielle de l'espace d'états et de l'interpolation : les très bonnes ou très mauvaises régions de l'espace d'états peuvent être découvertes tard dans les itérations (ou ne pas être découvertes du tout).

Deuxièmement, on peut remarquer qu'online-ATPI trouve rapidement (en moins de dix itérations) une politique qui rend le métro rentable. Il existe donc une telle politique malgré les contraintes économiques dures imposées au problème (la meilleure politique "à la main" testée avait une valeur de zéro pour s_0).

Enfin, ce même résultat illustre la principale faiblesse de cette première version de l'algorithme. Après avoir trouvé une bonne politique π_8, la valeur de l'état initial s'effondre à l'itération suivante ce qui est en contradiction à l'objectif de monotonie de l'évolution de la fonction de valeur. Ce phénomène est duû au fait qu'hors de la région explorée par la simulation à la dernière itération (hors de la région où les échantillons sont denses), la SVR surestime la valeur espérée à cause de l'absence d'échantillons dans ces régions (puisque la dernière bonne politique a justement évité ces zones). Par conséquent, lors de l'étape d'amélioration suivante, à chaque fois que le calcul d'une Q-valeur demande la valeur espérée en un état hors de la dernière zone explorée, cette valeur est trop élevée et la SVR n'agit plus comme une heuristique admissible (elle entraîne la politique vers les zones de faibles récompenses). A partir de cette politique biaisée, le processus d'amélioration reprend, comme on le voit aux itérations 9 à 14. Le paragraphe 5 présente une modification de l'algorithme pour résoudre ce problème.

5 Eviter la dégradation de la politique

Afin de conserver l'intérêt d'online-ATPI et la monotonie de l'évolution de la fonction de valeur, nous devons pouvoir garantir que nous ne surestimerons pas la valeur espérée d'un état pour une politique donnée. L'idée principale que nous développons consiste à garder trace des optimisations précédentes de la politique, en particulier sur les états qui ne sont plus parcourus par la simulation, en définissant des régions de confiance dans l'espace d'état relatives aux différentes étapes de l'optimisation.

5.1 Conserver une trace de la politique globale

Afin de ne pas "oublier" les optimisations passées, nous conservons un historique des actions optimisées au cours des itérations. En pratique, nous construisons un classifieur portant sur les paires (s, a) (un régresseur dans le cas des actions continues) qui généralise les actions trouvées à leur voisinage. Cela soulève deux problèmes : d'une part il faut pouvoir décider quand remplacer une action par une autre dans le classifieur, d'autre part il faut préciser la notion de voisinage afin de décider quand utiliser la politique optimisée à la place de la politique initiale. Nous apportons une réponse unique à ces deux problèmes en définissant une fonction indiquant où nous devons nous fier à notre interpolation ou non. En d'autres termes, nous avons besoin de définir une *fonction de confiance* qui indique si le classifieur a reçu suffisamment d'échantillons autour de l'état courant pour apprendre une action pertinente du point de vue statistique et fournir une solution raisonnable. Cette notion de confiance est cruciale pour l'algorithme amélioré d'online-ATPI que nous développerons plus en détail au paragraphe suivant. Finalement la modification de l'algorithme consiste à maintenir une base d'échantillons $actionDB$ composée de $x = (s, a)$, conservant ainsi trace de tous les états explorés. A l'itération n, les N_0 simulations fournissent un nouveau jeu d'actions optimisées sur le sous-espace d'état traversé par la simulation $actionDB'$. Nous retirons d'$actionDB$ tout échantillon x tel que $confidence(actionDB', x) = true$, rajoutons les échantillons d'$actionDB'$ dans $actionDB$ et enfin réentraînons le classifieur $\tilde{\pi}$ sur cette base d'apprentissage mise à jour. En pratique, nous utilisons des classifieurs à vecteurs supports pour les mêmes raisons que nous avons avancées pour l'usage des SVR. Pendant la simulation, nous utilisons la fonction de confiance de nouveau afin de déterminer si une action a été visitée à la dernière itération. Si c'est le cas, nous utilisons $\tilde{\pi}$, sinon π_0.

5.2 Quand faire confiance à l'évaluation par SVR de la fonction de confiance ?

Toutefois, conserver trace des optimizations précédentes ne nous garantit pas de ne pas sur-estimer la valeur espérée dans les régions mal explorées. Losqu'une étape d'amélioration nous entraîne loin des états explorés à la dernière itération, la SVR forunit une valeur qui tend (lorsqu'on s'éloigne des zones explorées) vers la moyenne des valeurs des points explorés. La fonction de confiance que nous avons définie au paragraphe précédent peut être utilisée ici pour définir la confiance dans l'estimation SVR. Nous fixons ainsi

un seuil sur la valeur de confiance en dessous duquel nous considérons la SVR comme non-fiable. Cela ne garantit pas que la SVR soit une heuristique admissible pour la zone de l'état courant mais garantit que sa valeur est proche de la fonction de valeur réelle. Afin de définir cette fonction de confiance sur la valeur, comme précédemment, nous construisons une base d'apprentissage $valueDB$ des échantillons (s, v) de la dernière itération. Ces échantillons correspondent à l'évaluation de la denrière politique calculée. Lorsque la fonction $ComputePolicy$ demande la valeur d'un état s', si $confidence(valueDB, s') = vrai$, alors nous utilisons la valeur de $\tilde{V}$. Sinon, nous effectuaons N_1 simulations à partir de s' en utilisant $\tilde{\pi}$ (ou π_0 si $\tilde{\pi}$ n'est pas fiable) afin d'enrichir la base $valueDB$ et d'adapter $\tilde{V}$. Nous arrêtons ces simulations dès qu'elles réentrent dans la zone de confiance sur la valeur (ce qui permet de réduire le temps de simulation en raccourcissant les trajectoires et le temps d'apprentissage en évitant les échantillons inutiles). Cela fournit une estimation de $V(s')$, met à jour la fonction globale $\tilde{V}$ et étend sa région de confiance associée.

On cherche à conserver la monotonie de l'évolution de la fonction de valeur en cas de régresseur parfait. Supposons que nous sommes en train d'effectuer l'itération $n + 1$. Nous voulons montrer que $V^{\pi_{n+1}}(s) \geq V^{\pi_n}(s)$. Par simplicité, nous négligeons l'erreur d'approximation dans la région de confiance. Si s est à l'intérieur de la région de confiance de $valueDB$, alors le résultat est immédiat. Si s est hors de cette région, alors nous pouvons décomposer tout chemin d'exécution commençant en s en un chemin hors zone e_{out} et une premier état à rencontrer de nouveau une zone de confiance s_{in}. Nous avons alors $V^{\pi_{n+1}}(s) = V(e_{out}) + V(s_{in})$, avec $V(e_{out}) = V^{\pi_n}(e_{out})$ et $V(s_{in}) \geq V^{\pi_n}(s_{in})$. Par conséquent $V^{\pi_{n+1}}(s) \geq V^{\pi_n}(s)$.

En pratique, il existe plusieurs différentes manières de définir la fonction de confiance. Nous nous sommes tournés vers les estimateurs de densité locaux et les SVM mono-classe, cependant d'autres travaux issus de l'apprentissage statistique pourraient s'appliquer ici. Ce sujet concerne encore les recherches futures. Néanmoins, il est rapide de noter qu'une fonction de confiance très sélective engendrera de très grandes bases d'apprentissage qui fourniront à leur tour une estimation très fine de la fonction de valeur et de la politique, tandis que des fonctions de confiance plus lâches réduiront le nombre de simulations à effectuer en déteriorant la qualité de l'estimateur. Ainsi, la fonction de confiance définit la granularité de nos fonctions de valeur et politiques. Enfin, dans le cadre ϵ et ν-SVR, les échantillons pour la régression arrivent incrémentalement ce qui oriente le choix du regrésseur vers des méthodes d'incremental batch SVR learning. Ce dernier point est important pour l'implantation d'*online-ATPI* car les premières expériences avec l'algorithme amélioré ont montré que le réentraînement ralentissait beaucoup le processus global d'apprentissage de la politique optimale. L'algorithme amélioré d'*online-ATPI* est présenté à l'algorithme 2.

6 Conclusion

Nous avons introduit une nouvelle méthode de recherche de politique sur des GSMDP à espace d'états de grande dimension. Notre méthode s'appuie sur l'exploration partielle de l'espace d'états, guidée par la politique courante, et vise à conserver dans la mesure du possible la monotonie d'évolution de la fonction de valeur d'un algorithme d'itération de la politique. Cette démarche permet de définir un algorithme spécifique d'itération de la politique temporelle approchée (ATPI) que nous nommons *online-ATPI*. Dans sa version courante, online-ATPI utilise extensivement les méthodes de SVR et SVC afin de généraliser des échantillons discrets à l'espace d'état continu ou hybride. Ce nouvel algorithme repose de plus sur une notion de granularité dans la recherche de la politque, au travers de l'usage de *fonctions de confiance* définissant la pertinence de l'usage des estimateurs SVM. Le travail en cours et futur s'axera sur l'évaluation plus en détail de l'algorithme amélioré.

Références

BELLMAN R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

BOYAN J. & LITTMAN M. (2001). Exact solutions to time dependent MDPs. *Advances in Neural Information Processing Systems*, **13**, 1026–1032.

CHANG C.-C. & LIN C.-J. (2001). *LIBSVM : a library for support vector machines*. Software available at `http://www.csie.ntu.edu./tw/~cjlin/libsvm`.

ERNST D., GEURTS P. & WEHENKEL L. (2005). Tree-based batch mode reinforcement learning. *JMLR*, **6**, 503–556.

FENG Z., DEARDEN R., MEULEAU N. & WASHINGTON R. (2004). Dynamic programming for structured continuous markov decision problems. In *20th Conference on Uncertainty in AI*, p. 154–161.

GLYNN P. (1989). A GSMP formalism for discrete event systems. *Proc. of the IEEE*, **77**.

HAUSKRECHT M. & KVETON B. (2006). Approximate linear programming for solving hybrid factord MDPs. In *9th Int. Symp. on AI and Math.*

Algorithm 2 Modified online-ATPI

```
main :
Input : π_0 or Ṽ_0, s_0
loop
  valueDB.reset(), newActionDB.reset()
  for i = 1 to N_0 do
    σ.reset()
    mainProcess.s ← s_0
    while horizon not reached do
      a =bestAction(s), activateEvent(a)
      (s', r) ← mainProcess.step(), σ.add(s, a, r)
    end while
    valueDB.convertExecutionPathToValueFunction(σ)
    newActionDB.add(σ)
  end for
  Ṽ ← TrainSVR(valueDB), π̃ ← TrainSVC(actionDB)
end loop

bestAction(s) :
for a ∈ A_s do
  Q̃(s, a) = 0
  for j = 1 to N_a do
    Q̃(s, a) = Q̃(s, a)+ simulateWithStop(s, a)
  end for
  Q̃(s, a) = Q̃(s,a) / N_a
  return arg max_{a∈A} Q̃(s, a)
end for

simulateWithStop(s, a) :
activateEvent(a), (s', r) ← mainProcess.clone().step()
Q ← r, s ← s'
while horizon not reached & confidence(s)=false do
  a = π̃(s), activateEvent(a)
  (s', r) ← mainProcess.clone().step()
  Q ← Q + r, s ← s'
end while
Q = Q + Ṽ(s)
valueDB.update(), Ṽ ← TrainSVR(valueDB)
return Q
```


LAGOUDAKIS M. & PARR R. (2003). Least-squares policy iteration. *JMLR*, **4**, 1107–1149.

MUNOS R. (2003). Error bounds for approximate policy iteration. In *Int. Conf. on Machine Learning*.

NIELSEN F. (1998). GMSim : a tool for compositionnal GSMP modeling. In *Winter Simulation Conference*.

ORMONEIT D. & SEN S. (2002). Kernel-based reinforcement learning. *Machine Learning*, **49**, 161–178.

PUTERMAN M. (1994). *Markov Decision Processes*. John Wiley & Sons, Inc.

QUESNEL G., DUBOZ R., RAMAT E. & TRAORE M. (2007). VLE - A Multi-Modeling and Simulation Environment. In *Moving Towards the Unified Simulation Approach, Proc. of the 2007 Summer Simulation Conf.*, p. 367–374.

RACHELSON E., FABIANI P., GARCIA F. & QUESNEL G. (2008a). Une approche basée sur la simulation pour l'optimisation des processus décisionnels semi-markoviens généralisés (english version). In *Conf. Francophone sur l'Apprentissage Automatique*.

RACHELSON E., GARCIA F. & FABIANI P. (2008b). Extending the bellman equation for MDP to continuous actions and continuous time in the discounted case. In *10th Int. Symp. on AI and Math.*

VAPNIK V., GOLOWICH S. & SMOLA A. (1996). Support vector method for function approximation, regression estimation and signal processing. *Advances in Neural Information Processing Systems*, **9**, 281–287.

WHITESON S. & STONE P. (2006). Evolutionary function approximation for reinforcement learning. *JMLR*, **7**, 877–917.

YOUNES H. & SIMMONS R. (2004). Solving generalized semi-markov decision processes using continuous phase-type distributions. In *AAAI*.

ZEIGLER B. P., KIM D. & PRAEHOFER H. (2000). *Theory of modeling and simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

Filtrage bayésien de la récompense

Matthieu Geist[1,2], Olivier Pietquin[1] et Gabriel Fricout[2]

[1] Supélec
Groupe de recherche IMS, Metz, France
{matthieu.geist,olivier.pietquin}@supelec.fr

[2] ArcelorMittal Research
Département MCE, Maizières-lès-Metz, France
gabriel.fricout@arcelormittal.com

Résumé :
Une large variété de schémas d'approximation de la fonction de valeur a été appliquée à l'apprentissage par renforcement. Cependant, les approches par filtrage bayésien, qui se sont pourtant montrées efficaces dans d'autres domaines comme l'apprentissage de paramètres pour les réseaux neuronaux, ont été peu étudiées jusqu'à présent. Cette contribution introduit un cadre de travail général pour l'apprentissage par renforcement basé sur le filtrage bayésien, ainsi qu'une implémentation spécifique basée sur un filtre de Kalman à sigma-points et une paramétrisation par noyaux. Cela nous permet de proposer un algorithme de différences temporelles pour des espaces d'état et/ou d'action continus qui soit *model-free* et *off-policy*. Il sera illustré sur deux problèmes simples.

Mots-clés : Apprentissage par renforcement, filtrage bayésien, méthodes à noyaux.

1 Introduction

Le contrôle optimal d'un système dynamique et stochastique peut être un problème très complexe. Même avec une connaissance parfaite du système physique, il peut être impossible de déterminer la politique de contrôle optimale de façon analytique. La réponse habituelle de l'industrie à ce type de problème est de faire appel à des heuristiques, qui reposent sur de fortes connaissances humaines *a priori* et pour lesquelles il est difficile de garantir l'optimalité. D'un autre côté, dans le domaine de l'apprentissage numérique, ce problème est traité par l'apprentissage par renforcement (AR) (Sutton & Barto, 1998; Bertsekas, 1995). Le problème de contrôle est alors décrit en terme d'états, d'actions et de récompenses. Dans ce cadre de travail, un agent artificiel essaye d'apprendre une politique de contrôle optimale à partir d'interactions avec son environnement. Il observe l'état du système et choisit une action à lui appliquer, en accord avec sa politique interne qui lie les états aux actions. En retour, l'agent reçoit un signal numérique de récompense, qui est une indication locale et instantanée de la qualité du contrôle. Cette information de récompense est utilisée par l'agent pour apprendre de façon incrémentale la politique de contrôle optimale qui maximisera une fonction du cumul futur des récompenses. Classiquement, la connaissance qu'a l'agent de l'environnement est modélisée par une Q-fonction qui associe à un couple état-action donné une estimation du cumul espéré de récompenses associé. La Q-fonction optimale (notée Q^*) lie chaque paire état-action au maximum possible de cumul de récompenses. L'action optimale est alors celle qui maximise cette fonction pour l'état courant, ce qui rend la connaissance de Q^* suffisante pour réaliser la tâche de contrôle. Le rôle de l'agent peut donc se résumer en l'apprentissage de cette fonction à partir d'interactions avec l'environnement. Cela sera décrit plus avant dans la section 2.1.

L'AR est basé sur un principe d'apprentissage par essai/erreur qui fait sa force mais également sa faiblesse. En fait, l'AR ne nécessite aucune connaissance *a priori* du système, mais suppose d'acquérir des informations sur ce dernier à travers des essais qui pourraient l'endommager. Cela est particulièrement vrai lorsque l'espace d'état est large, étant donné qu'il reste alors majoritairement inconnu au début de l'apprentissage. En pratique, il arrive souvent que l'espace d'état soit trop large (par exemple continu), ce qui l'empêche d'être exploré de façon exhaustive par l'agent. Proposer des algorithmes capables de prendre en compte de tels espaces tout en conservant un apprentissage incrémental à partir d'interactions est ainsi devenu un défi pour la communauté de l'apprentissage numérique. D'autre part, l'AR consiste également à

apprendre à agir dans le doute. L'incertitude provient de la nature stochastique du système à contrôler, mais également de la connaissance partielle qu'a l'agent de son environnement. Cette connaissance partielle a au moins deux origines distinctes : soit l'espace d'état n'a pas été suffisamment exploré par l'agent, soit l'état lui même n'est pas directement observable (observabilité partielle). Cela rend le processus d'apprentissage encore plus difficile, et l'agent doit constamment choisir entre deux types d'action : les actions exploitatives, qui sont optimales respectivement à la connaissance qu'a l'agent du système, et les actions exploratoires, qui ont pour but d'améliorer cette connaissance. Ce problème est connu sous le nom de dilemme entre exploration et exploitation. La généralisation, l'observabilité partielle et le dilemme entre exploration et exploitation sont actuellement les domaines de recherches les plus importants en apprentissage par renforcement.

Classiquement, dans le domaine de l'intelligence artificielle, l'incertitude est prise en compte à l'aide de méthodes bayésiennes. Ces dernières sont adaptées à un apprentissage en ligne étant donné qu'elles sont généralement incrémentales. Le filtrage bayésien peut par exemple être utilisé pour apprendre les paramètres d'une fonction approximant une fonction d'intérêt tout en maintenant une incertitude liée à la régression (voir par exemple van der Merwe, 2004). Nous présentons le filtrage bayésien section 2.2. Bien que ces méthodes de filtrage aient déjà été appliquées à l'apprentissage par renforcement, elles ont jusqu'à présent été étonnamment peu étudiées. Dans cette contribution, nous proposons les prémisses d'une méthode basée sur le filtrage Bayésien et pouvant traiter les problèmes majeurs de l'apprentissage par renforcement, et particulièrement celui de la généralisation qui sera le seul traité ici. Le reste de cet article est organisé comme suit. Dans un premier temps quelques notions de bases sont présentées. Puis la formulation générale de l'apprentissage par renforcement comme un problème de filtrage bayésien ainsi qu'une implémentation spécifique sont proposées. Enfin nous montrons les premiers résultats concernant deux problèmes simples, le *wet-chicken* et le *mountain-car*, avant de conclure et d'esquisser nos travaux futurs.

2 Contexte

Dans cette section, nous présentons le formalisme mathématique classiquement associé à l'apprentissage par renforcement et l'algorithme du Q-learning, ainsi que le paradigme du filtrage bayésien et ses solutions spécifiques. Par la suite, une variable x servira à désigner un vecteur colonne ou un scalaire, ce qui devrait apparaître clairement d'après le contexte.

2.1 Apprentissage par renforcement

Un processus décisionnel de Markov (PDM) consiste en un espace d'état S, un espace d'action A, une probabilité de transition markovienne $p : S \times A \rightarrow \mathcal{P}(S)$ et une fonction de récompense bornée $r : S \times A \times S \rightarrow \mathbb{R}$. Une politique est une fonction qui à un état associe une action : $\pi : S \rightarrow A$. Au temps k, le système est dans un état s_k, l'agent choisit une action $a_k = \pi(s_k)$, et le système passe alors dans un nouvel état s_{k+1} selon la distribution conditionnelle markovienne $p(.|s_k, a_k)$. L'agent reçoit alors la récompense associée $r_k = r(s_k, a_k, s_{k+1})$. Son objectif est de trouver la politique qui maximise l'espérance du cumul des récompenses pondérées, c'est-à-dire la quantité $E_\pi[\sum_{k\in\mathbb{N}} \gamma^k r(S_k, A_k, S_{k+1})|S_0 = s_0]$, pour tout état initial possible s_0, l'espérance étant calculée sur les transitions d'états, la politique π étant suivie, et $\gamma \in [0, 1[$ étant un facteur d'oubli permettant de tenir plus ou moins compte de la valeur des états futurs.

Une approche classique pour résoudre ce problème est d'introduire une fonction Q définie par :

$$Q^\pi(s, a) = \int_S p(z|s, a)\Big(r(s, a, z) + \gamma Q^\pi(z, \pi(z))\Big)dz$$

C'est le cumul espéré de récompenses en choisissant l'action a dans l'état s puis en suivant la politique π par la suite. Le critère d'optimalité est de trouver la politique π^* (et la fonction Q^* associée) telle que pour tout état s et pour toute politique π, on ait $\max_{a\in A} Q^*(s, a) \geq \max_{a\in A} Q^\pi(s, a)$. La Q-fonction optimale Q^* est donnée par l'équation de Bellman :

$$Q^*(s, a) = \int_S p(z|s, a) \left(r(s, a, z) + \gamma \max_{b\in A} Q^*(z, b)\right) dz$$

La politique optimale s'en déduit aisément, c'est la politique gloutonne associée : $\forall s \in S$, $\pi^*(s) = \mathrm{argmax}_{a\in A} Q^*(s, a)$. Dans le cas d'espaces d'état et d'action discrets et finis, l'algorithme du Q-learning

fournit une solution à ce problème. Son principe est de mettre à jour une approximation tabulaire de la Q-fonction optimale après chaque transition (s, a, r, s') :

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left(r + \gamma \max_{b \in A} \hat{Q}(s',b) - \hat{Q}(s,a) \right)$$

où α est un taux d'apprentissage. Un fait intéressant est que le Q-learning est un algorithme *off-policy*, c'est-à-dire qu'il permet d'apprendre la politique optimale (via la Q-fonction associée) en en suivant une sous-optimale, si cette dernière est suffisamment exploratoire. La présente contribution peut être vue comme une extension de cet algorithme dans un cadre bayésien (avec d'autres avantages). Nous renvoyons le lecteur à Sutton & Barto (1998) pour une introduction plus complète à l'apprentissage par renforcement.

2.2 Filtrage Bayésien

Le problème du filtrage bayésien peut s'exprimer sous sa forme espace d'état :

$$\begin{aligned} x_{k+1} &= f_k(x_k, v_k) \\ y_k &= g_k(x_k, n_k). \end{aligned}$$

L'objectif est d'inférer séquentiellement l'état caché x_k sachant les observations $y_{1:k} = \{y_1, y_2, \ldots, y_k\}$. L'évolution de l'état se fait selon la (potentiellement non-linéaire et non-stationnaire) fonction f_k est le bruit d'évolution v_k. L'observation y_k est une fonction (également potentiellement non-linéaire et non-stationnaire) de l'état x_k, corrompue par une bruit d'observation n_k. Le principe général est de prédire le nouvel état x_k étant donné les précédentes observations $y_{1:k-1}$, et de corriger cette prédiction en utilisant la nouvelle observation y_k, en accord avec les équations dites de prédiction et de correction (utilisant la loi de Bayes) :

$$\begin{aligned} p(X_k|Y_{1:k-1}) &= \int_{\mathcal{X}} p(X_k|X_{k-1}) p(X_{k-1}|Y_{1:k-1}) dX_{k-1} \text{ (prédiction)}, \\ p(X_k|Y_{1:k}) &= \frac{p(Y_k|X_k) p(X_k|Y_{1:k-1})}{\int_{\mathcal{X}} p(Y_k|X_k) p(X_k|Y_{1:k-1}) dX_k} \text{ (correction)}. \end{aligned}$$

Ces équations ne sont généralement pas résolubles. Si les fonctions sont linéaires et les bruits gaussiens, la solution optimale est donnée par le filtre de Kalman : les quantités d'intérêt sont des variables aléatoires, et l'inférence (c'est-à-dire la prédiction de ces quantités et leur correction étant donné une nouvelle observation) est faite en ligne par propagation des statistiques suffisantes à travers des transformations linéaires. Si les fonctions sont non-linéaires (et si les bruits sont toujours gaussiens), une solution est de les linéariser autour de l'état : c'est le principe du filtrage de Kalman étendu (ou FKE), les statistiques d'intérêt sont toujours propagées à travers des transformations linéaires. Une autre approche est l'ensemble des filtres de Kalman à sigma-points, ou FKSP (van der Merwe, 2004). L'idée sous-jacente à ce modèle est qu'il est plus facile d'approximer une distribution de probabilité qu'une fonction non-linéaire arbitraire. Un ensemble de points (appelés sigma-points) est calculé de façon déterministe à partir des statistiques de premier et deuxième ordre de l'état caché. Ils sont représentatifs de la distribution d'intérêt. Les images de ces points par les fonctions d'évolution et d'observation sont calculées, et elles sont utilisées pour déterminer les statistiques nécessaires pour les équations de prédiction et de correction. L'algorithme 1 esquisse une mise à jour d'un FKSP dans le cas de bruits additifs, où des notations standards sont utilisées : $x_{k|k-1}$ désigne une prédiction, $x_{k|k}$ une estimée (ou une correction), $P_{x,y}$ la matrice de covariance des vecteurs aléatoires x et y, $\bar{n}_k$ une moyenne et k est l'index temporel. Le lecteur peut se référer à van der Merwe (2004) pour plus de détails. C'est le type de filtre que nous utilisons dans l'implémentation spécifique proposée. Une dernière méthode est le filtrage particulaire, ou Monte Carlo séquentiel. C'est une approche numérique adaptée au cas où les fonctions sont non-linéaires et les bruits non-gaussiens. Nous renvoyons le lecteur à Chen (2003) pour une vue d'ensemble complète des différentes méthodes de filtrage bayésien.

3 Formulation générale

Maintenant que l'apprentissage par renforcement et le filtrage bayésien ont été présentés, nous pouvons introduire le canevas du filtrage bayésien de la récompense. Nous commençons par présenter l'idée générale ainsi que des travaux similaires, avant de présenter plus formellement le cadre de travail général proposé.

Algorithme 1 Mise à jour d'un FKSP

Entrées : $x_{k-1|k-1}$, $P_{k-1|k-1}$
Sorties : $x_{k|k}$, $P_{k|k}$
Calcul des sigma-points :
Calculer de façon déterministe l'ensemble des sigma-points $X_{k-1|k-1}$ à partir de $x_{k-1|k-1}$ et $P_{k-1|k-1}$;
Etape de prédiction :
Calculer $X_{k|k-1}$ à partir de $f_k(X_{k-1|k-1}, \bar{v}_k)$ et de la covariance du bruit d'évolution ;
Calculer $x_{k|k-1}$ et $P_{k|k-1}$ à partir de $X_{k|k-1}$;
Etape de Correction :
Observer y_k ;
$Y_{k|k-1} = g_k(X_{k|k-1}, \bar{n}_k)$;
Calculer $y_{k|k-1}$, $P_{y_{k|k-1}}$ et $P_{x_{k|k-1},y_{k|k-1}}$ à partir de $X_{k|k-1}$, $Y_{k|k-1}$ et de le covariance du bruit d'observation ;
$K_k = P_{x_{k|k-1},y_{k|k-1}} P^{-1}_{y_{k|k-1}}$; {*gain de Kalman*}
$x_{k|k} = x_{k|k-1} + K_k(y_k - y_{k|k-1})$;
$P_{x_{k|k}} = P_{x_{k|k-1}} - K_k P_{y_{k|k-1}} K_k^T$;

3.1 Idée générale et travaux similaires

L'idée générale est de paramétrer la Q-fonction, et de considérer le vecteur de paramètres associé comme étant l'état caché que devra inférer un filtre bayésien. L'équation d'observation associée lie les récompenses aux paramètres grâce à l'équation de Bellman (1957). L'idée d'utiliser le filtrage bayésien pour résoudre des problèmes d'apprentissage par renforcement est déjà apparue dans quelques publications. Engel (2005) utilise des processus Gaussiens pour l'apprentissage par renforcement. Son approche peut être vue comme une extension du filtre de Kalman à un vecteur d'état de dimension infinie (le processus Gaussien), cependant l'équation d'observation est nécessairement linéaire, et contrairement à cette contribution seuls des algorithmes *on-policy* sont possibles, ce qui impose d'adopter des schémas du type itération de la politique. Dans (Phua & Fitch, 2007) un banc de filtres de Kalman est utilisé pour déterminer une paramétrisation linéaire par morceau de la fonction de valeur. Cette approche peut être vue comme un cas spécifique (bien que non trivial) du canevas général proposé ici.

3.2 Filtrage bayésien de la récompense

L'équation de Bellman peut s'écrire :

$$Q^*(s,a) = r(s,a,s') + \gamma \max_{b \in A} Q^*(s',b) - n_{s,a}(s')$$

$$\text{avec } n_{s,a}(s') = \int_S p(z|s,a) \left\{ r(s,a,s') - r(s,a,z) + \gamma \left(\max_{b \in A} Q^*(s',b) - \max_{b \in A} Q^*(z,b) \right) \right\} dz.$$

Le terme de bruit $n_{s,a}$ est la transformation non linéaire de la variable aléatoire $(S'|S = s, A = a)$ de loi $p(.|s,a)$, c'est donc une variable aléatoire. Il peut être montré qu'elle est centrée et de variance finie (d'après l'inégalité de Minkowsky), c'est-à-dire, r_{max} étant la borne sur la fonction de récompense :

$$E[n_{s,a}] = \int_S n_{s,a}(z) p(z|s,a) dz = 0 \quad \text{et} \quad E[n_{s,a}^2] = \int_S n_{s,a}^2(z) p(z|s,a) dz \leq \left(\frac{r_{max}}{1-\gamma} \right)^2$$

Pour une transition (s,a,r,s') observée, l'équation de Bellman peut donc être réécrite :

$$r(s,a,s') = Q^*(s,a) - \gamma \max_{b \in A} Q^*(s',b) + n_{s,a}(s')$$

Cette forme est similaire à la mise à jour de l'algorithme du Q-learning, et est de première importance pour le cadre de travail proposé.

Supposons que la Q-fonction est paramétrée par un vecteur θ, linéairement ou non. L'objectif est de déterminer une bonne approximation $\hat{Q}_\theta$ de la Q-fonction optimale Q^* à partir de l'observation de transitions

(s, a, r, s'). Ce problème de régression par la récompense est exprimé sous une forme espace d'état. Pour une transition (s_k, a_k, r_k, s'_k), il s'écrit :

$$\begin{aligned}\theta_{k+1} &= \theta_k + v_k \\ r_k &= \hat{Q}_{\theta_k}(s_k, a_k) - \gamma \max_{a \in A} \hat{Q}_{\theta_k}(s'_k, a) + n_k.\end{aligned}$$

Ici v_k est un bruit d'évolution artificiel et n_k est un bruit d'observation centré incluant toute la stochasticité du PDM (ainsi que l'erreur de modélisation). La formulation générale est posée, mais elle est encore loin d'être résolue. L'équation d'observation est non-linéaire (à cause de l'opérateur $\max$), c'est pourquoi les filtres classiques comme celui de Kalman ne peuvent être utilisés. Formellement, le bruit d'évolution est nul, mais introduire un bruit artificiel d'évolution peut permettre d'améliorer la stabilité et la convergence du filtre (par exemple en aidant à éviter les optima locaux). Un bruit d'observation, potentiellement adaptatif, doit être choisi, ainsi qu'une paramétrisation pour la Q-fonction. Enfin, comme pour toute approche bayésienne, un *a priori* sur les paramètres doit être fixé pour initialiser l'algorithme itératif. Une implémentation spécifique d'un filtre bayésien de la récompense est proposée dans la section suivante.

4 Une solution pratique

Les deux plus grands degrés de liberté sont le choix de la paramétrisation et du filtre. Nous choisissons une paramétrisation par noyaux pour la Q-fonction, que nous rendons parcimonieuse à l'aide d'une méthode de dictionnaire. L'équation d'observation contenant l'opérateur $\max$, le filtre doit être non-linéaire et ne pas nécessiter de calcul de dérivées. Nous choisissons d'utiliser un filtre à sigma-points.

4.1 Paramétrisation

Une paramétrisation par noyaux est choisie pour la Q-fonction, en raison de son expressivité donnée par le théorème de Mercer (Vapnik, 1998). Un noyaux est une fonction continue, symétrique et définie positive. Chaque noyau correspond à un produit scalaire dans un espace de (généralement) plus grande dimension. Plus précisément, pour chaque noyau K, il existe une fonction φ de l'espace de départ $\mathcal{X}$ dans l'espace dit des caractéristiques $\mathcal{F}$ tel que $\forall x, y \in \mathcal{X}$, $K(x, y) = \langle \varphi(x), \varphi(y) \rangle$. Cette propriété fondamentale est importante pour l'initialisation de notre algorithme. Plus précisément, des noyaux gaussiens sont choisis, et leurs centres, leurs variances ainsi que les poids associés sont paramétrés :

$$\hat{Q}_\theta(s, a) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i^s}(s, s_i) K_{\sigma_i^a}(a, a_i) \text{ avec } K_{\sigma_i^x}(x, x_i) = \exp\left(-(x - x_i)^T (\Sigma_i^x)^{-1} (x - x_i)\right),$$
$$\text{où } x = s, a,\ \Sigma_i^x = \mathrm{diag}(\sigma_i^x)^2, \text{ et } \theta = [(\alpha_i)_{i=1}^p, (s_i^T)_{i=1}^p, (a_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p, ((\sigma_i^a)^T)_{i=1}^p]^T$$

L'opérateur diag appliqué à un vecteur colonne produit une matrice diagonale. Notons que le noyau utilisé $K_{(\sigma_i^s, \sigma_i^a)}([s^T, a^T]^T, [s_i^T, a_i^T]^T) = K_{\sigma_i^s}(s, s_i) K_{\sigma_i^a}(a, a_i)$ est un produit de noyaux gaussiens, c'est donc un noyau gaussien.

4.2 Dictionnaire

Un premier problème est de choisir le nombre p de fonctions noyaux et l'*a priori* associé. Pour cela, nous posons un *a priori* sur la largeur des gaussiennes : $\sigma_0^T = [(\sigma_0^s)^T, (\sigma_0^a)^T]$. Une méthode de dictionnaire proposée notamment dans (Engel, 2005) est ensuite utilisée pour déterminer le nombre de noyaux et l'*a priori* sur leurs centres. Considérons un noyau K, la fonction associée φ et un ensemble de points $X = \{x_1, x_2, \ldots\}$. L'objectif de cette méthode est de calculer un dictionnaire de p points $\mathcal{D} = \{\tilde{x}_1, \ldots, \tilde{x}_p\}$ tel que $\varphi(\mathcal{D})$ soit une base approximative de $\varphi(X)$ qui est inclus dans l'espace des caractéristiques associé au noyau K.

Cette procédure est itérative. Supposons que les échantillons $x_1, x_2, \ldots$ sont séquentiellement observés (dans notre cas particulier, les échantillons sont des couples état-action). Au temps k, un dictionnaire $\mathcal{D}_{k-1} = (\tilde{x}_j)_{j=1}^{m_{k-1}} \subset (x_j)_{j=1}^{k-1}$ est disponible où par construction les vecteurs caractéristiques $\varphi(\tilde{x}_j)$ sont approximativement linéairement indépendants dans $\mathcal{F}$. L'exemple x_k est ensuite observé, et est ajouté au

dictionnaire si $\varphi(x_k)$ est approximativement linéairement indépendant de $\mathcal{D}_{k-1}$. Pour tester cela, des poids $w = (w_1, \ldots, w_{m_{k-1}})^T$ doivent être calculés de façon à vérifier

$$\delta_k = \min_w \left\| \sum_{j=1}^{m_{k-1}} w_j \varphi(\tilde{x}_j) - \varphi(x_k) \right\|^2$$

Un seuil prédéfini ν, qui sert à déterminer la qualité de l'approximation (et en conséquence la parcimonie du dictionnaire), est utilisé. Si $\delta_k > \nu$, $x_k = \tilde{x}_{m_k}$ est ajouté au dictionnaire, sinon $\varphi(x_k)$ s'écrit :

$$\varphi(x_k) = \sum_{i=1}^{m_{k-1}} w_i \varphi(\tilde{x}_i) + \varphi_{res} \text{ avec } \|\varphi_{res}\|^2 \leq \nu$$

Nous définissons la matrice $\tilde{K}_{k-1}$ de taille $m_{k-1} \times m_{k-1}$ et le vecteur $\tilde{k}_{k-1}(x)$ de taille $m_{k-1} \times 1$ par :

$$\left(\tilde{K}_{k-1}\right)_{i,j} = K(\tilde{x}_i, \tilde{x}_j) \text{ et } \left(\tilde{k}_{k-1}(x)\right)_i = K(x, \tilde{x}_i)$$

En utilisant la bilinéarité du produit scalaire et en remplaçant ces derniers par les noyaux, δ_k peut s'exprimer de la façon suivante :

$$\delta_k = \min_w \left\{ w^T \tilde{K}_{k-1} w - 2w^T \tilde{k}_{k-1}^T(x_k) + K(x_k, x_k) \right\}$$

La solution de ce problème est $\delta_k = K(x_k, x_k) - \tilde{k}_{k-1}(x_k) w_k$ avec $w_k = \tilde{K}_{k-1}^{-1} \tilde{k}_{k-1}(x_k)$. De plus il existe une version efficace de cet algorithme utilisant le lemme d'inversion matricielle (qui permet de calculer directement l'inverse, de façon incrémentale). Le lecteur peut se référer à Engel (2005) pour plus de détails. Pratiquement nous supposerons que S et A sont des ensembles compacts et que leur bornes sont connues. Le dictionnaire associé peut alors être calculé dans une phase de prétraitement à partir de N échantillons générés uniformément à partir de $S \times A$.

4.3 *A priori* gaussien

Comme pour chaque approche bayésienne, un *a priori* doit être choisi pour la distribution des paramètres. Nous posons $\theta_0 \sim \mathcal{N}(\bar{\theta}_0, \Sigma_{\theta_0})$ où

$$\bar{\theta}_0 = [\alpha_0, \ldots, \mathcal{D}_s, \mathcal{D}_a, (\sigma_0^s)^T, \ldots, (\sigma_0^a)^T, \ldots]^T$$
$$\Sigma_{\theta_0} = \text{diag}(\sigma_{\alpha_0}^2, \ldots, \sigma_{\mu_0^s}^2, \ldots, \sigma_{\mu_0^a}^2, \ldots, \sigma_{\sigma_0^s}^2, \ldots, \sigma_{\sigma_0^a}^2, \ldots)$$

Dans ces expressions, α_0 est la moyenne *a priori* sur les poids des noyaux, $\mathcal{D} = \mathcal{D}_s \times \mathcal{D}_a$ est l'ensemble des moyennes *a priori* des centres des noyaux, calculé en prétraitement par la méthode de dictionnaire à partir de la moyenne *a priori* de l'écart-type des noyaux gaussiens $\sigma_0^T = [(\sigma_0^s)^T, (\sigma_0^a)^T]$, et $\sigma_{\alpha_0}^2$, $\sigma_{\mu_0^x}^2$, $\sigma_{\sigma_0^x}^2$ sont respectivement les variances *a priori* sur les poids, les centres et les écarts-type des noyaux, pour $x = s, a$. Tous ces paramètres (excepté le dictionnaire) doivent être définis à l'avance, en utilisant les connaissances que l'on peut avoir du problème d'intérêt. Soit $q = (2(n_a + n_s) + 1)\, p$, où n_s (respectivement n_a) est la dimension de l'espace d'état (respectivement de l'espace d'action). Notons que $\bar{\theta}_0 \in \mathbb{R}^q$ et $\Sigma_{\theta_0} \in \mathbb{R}^{q \times q}$. Nous posons également l'*a priori* sur les bruits : $v_0 \sim \mathcal{N}(0, R_{v_0})$ et $n_0 \sim \mathcal{N}(0, R_{n_0})$, où $R_{n_0} = \sigma_{n_0}^2$.

4.4 Mise à jour des paramètres

Une fois que les paramètres sont initialisés, il reste toujours à les mettre à jour quand de nouvelles observations (s_k, a_k, r_k, s'_k) sont disponibles. Pour cela, nous utilisons un FKSP spécifique, un SR-CDKF (*Square-Root Central Difference Kalman Filter*). Le lecteur peut se référer à van der Merwe (2004) pour une description plus précise. Un dernier problème est de choisir un bruit artificiel d'évolution. Formellement, comme la fonction cible est stationnaire, il est nul. Cependant, son introduction peut améliorer la stabilité et la convergence du filtre.

La covariance du bruit d'évolution est choisie comme étant une fraction de la covariance des paramètres :

$$R_{v_k} = (\lambda^{-1} - 1)\Sigma_{\theta_k}$$

où $\lambda \in]0, 1]$ $(1 - \lambda \ll 1)$ est un facteur d'oubli similaire à celui utilisé dans l'algorithme des moindres carrés récursifs. Dans ce papier, nous choisissons un bruit d'observation constant, c'est-à-dire que $R_{n_k} = R_{n_{k-1}}$. Le filtre bayésien de la récompense proposé est résumé dans l'algorithme 2.

Algorithme 2 Un filtre bayésien de la récompense

entrées : ν, N, α_0, σ_0, σ_{α_0}, $\sigma_{\mu_0^x}$, $\sigma_{\sigma_0^x}$, σ_{v_0}, σ_{n_0}
sorties : $\bar{\theta}$, Σ_θ
Calcul du dictionnaire :
$\forall i \in \{1 \dots N\},\ [s_i^T, a_i^T]^T \sim \mathcal{U}_{\mathcal{S}\times\mathcal{A}}$;
Poser $X = \{[s_1^T, a_1^T]^T, \dots, [s_N^T, a_N^T]^T\}$;
$\mathcal{D}$ =`Compute-Dictionary`(X,ν,σ_0)
Initialisation :
Initialiser $\bar{\theta}_0$, Σ_{θ_0}, R_{n_0}, R_{v_0} ;
pour $k = 1, 2, \dots$ **faire**
  Observer $t_k = (s_k, a_k, r_k, s'_k)$;
  Mise à jour du SR-CDKF :
  $[\bar{\theta}_k, \Sigma_{\theta_k}, K_k]$ = `SR-CDKF-Update`($\bar{\theta}_{k-1}$,$\Sigma_{\theta_{k-1}}$,t_k,$R_{v_{k-1}}$,$R_{n_{k-1}}$) ;
  Mise à jour du bruit artificiel d'évolution :
  $R_{v_k} = (\lambda^{-1} - 1)\Sigma_{\theta_k}$;
fin pour

4.5 Maximum sur l'espace d'action

Une difficulté technique est le calcul du maximum de la Q-fonction paramétrée sur l'espace d'action. Ce calcul est nécessaire pour la mise à jour du filtre. Avec la paramétrisation adoptée, cela n'est pas résoluble analytiquement. Une première solution serait d'échantillonner l'espace d'état et de calculer le maximum des échantillons obtenus. Cependant ce n'est pas très efficace. La méthode utilisée est proche de celle proposée par Carreira-Perpinan (2000).

Le maximum sur les centres des noyaux actions est d'abord calculé : $\mu^a = \mathrm{argmax}_{a_i} \hat{Q}_\theta(s, a_i)$. Il sert d'initialisation pour la méthode de Newton-Raphson utilisée pour trouver le maximum :

$$a_m \leftarrow a_m - \Big((\nabla_a \nabla_a^T)\hat{Q}_\theta(s,a)\Big)^{-1}_{a=a_m} \nabla_a \hat{Q}_\theta(s,a)\Big|_{a=a_m}$$

Lorsque la hessienne est singulière, une méthode de type descente de gradient/point fixe est utilisée :

$$a_m \leftarrow a_m + \nabla_a \hat{Q}_\theta(s,a)\Big|_{a=a_m}$$

L'action a_m obtenue est considérée comme l'action qui maximise la Q-fonction paramétrée.

5 Résultats préliminaires

L'approche proposée est testée sur deux problèmes. Le premier, le *wet-chicken*, est une tâche stochastique à espaces d'état et d'action continus. Le second, le *mountain car* est une tâche déterministe, à espace d'action discret et à espace d'état continu. Ces choix simples sont fait afin de démontrer que la méthode peut s'adapter à différentes structures de problèmes. Nous adopterons une paramétisation hybride pour le second. Mais nous discutons dans un premier temps le choix des paramètres.

5.1 Choix des paramètres

Pour les deux problèmes le facteur γ est fixé à 0.9. Le facteur de parcimonie du dictionnaire a été fixé à une valeur assez importante, $\nu = 0.9$. Le facteur d'oubli associé au bruit artificiel d'évolution a également été fixé à une haute valeur : $\lambda^{-1} - 1 \simeq 10^{-6}$.

Le choix initial des écarts-type des noyaux est dépendant du problème. Cependant, en pratique, un bon choix semble être de prendre une fraction de la quantité $x(j)_{max} - x(j)_{min}$ pour l'écart-type associé à $x(j)$, la $j^{\text{ème}}$ composante du vecteur x, $x(j)_{max}$ et $x(j)_{min}$ étant les bornes sur les valeurs prises par $x(j)$. Nous supposons l'*a priori* sur les poids des noyaux centrés, et nous posons l'écart-type associé comme étant une faible fraction de la borne théorique de la Q-fonction $\frac{r_{max}}{1-\gamma}$. En raison de la géométrie des noyaux gaussiens, nous supposons que les centres fournis par le dictionnaire sont approximativement uniformément

distribués, et nous posons l'écart-type de la $j^{\text{ème}}$ composante du vecteur x comme étant une faible fraction de $(x(j)_{max} - x(j)_{min})p^{-\frac{1}{ns+na}}$, avec la convention que pour les espaces discrets $n = 0$. Finalement, nous posons l'écart-type de l'*a priori* associé à l'écart-type des noyaux comme étant une faible fraction de ces derniers. Autrement dit, nous posons $\sigma_{\sigma_0^{x(j)}}$ comme étant une petite fraction de $\sigma_0^{x(j)}$, pour la $j^{\text{ème}}$ composante de x.

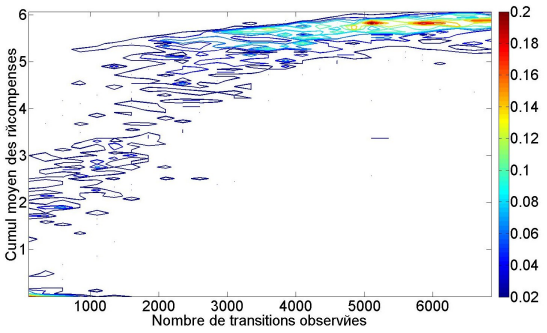


FIG. 1 – Histogramme 2d du cumul moyen de récompenses pour le *wet-chicken* (100 *runs*).

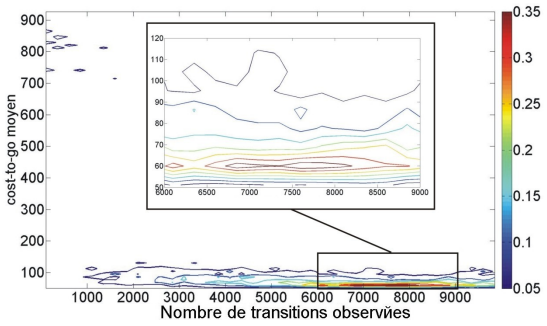


FIG. 2 – Histogramme 2d du *cost-to-go* moyen pour le problème du *mountain-car* (100 *runs*).

5.2 Problème du *wet-chicken*

Dans le problème du *wet-chicken* (inspiré de Schneegass *et al.*, 2006), un canoéiste doit pagayer sur une rivière jusqu'à atteindre une chute d'eau. La récompense augmente linéairement avec la proximité de la chute. Le canoéiste reçoit une punition s'il tombe, et il doit alors recommencer. Des turbulences simulées rendent le problème stochastique. Plus formellement, l'espace d'état est $S = [0, 10]$ (10 étant la position de la chute), l'espace d'action est $A = [-1, 1]$ (continûment de pagayer complètement vers l'arrière à pagayer complètement vers l'avant), la transition est $s' = s + a + c$ avec $c \sim \mathcal{N}(0, \sigma_c)$, $\sigma_c = 0.3$, et la récompense associée est $r = \frac{s'}{10}$. Si $s' \geq 10$ le canoéiste tombe, la récompense est $r = -1$ et l'épisode est terminé.

Pour tester l'algorithme proposé, des transitions aléatoires sont générées uniformément et présentées à l'entrée du filtre : à chaque pas de temps k, un état s_k et une action a_k sont générées uniformément à partir de $S \times A$, et utilisés pour tirer une transition (aléatoire) vers s'_k, avec la récompense associée r_k, et la transition (s_k, a_k, s'_k, r_k) est présentée en entrée de l'algorithme. Les résultats sont présentés figure 1. Pour chaque *run* de l'algorithme et toutes les 250 transitions observées, le cumul espéré de récompenses pour la politique courante a été calculé comme le cumul moyen de récompenses sur 1000 épisodes, chacun étant initialisé aléatoirement (distribution uniforme). La moyenne est donc faite sur la stochasticité des transitions et de l'état de départ. Notons que la durée de vie de l'agent (la durée maximum d'un épisode) est bornée à 100 interactions avec l'environnement. Nous avons ensuite calculé l'histogramme en deux dimensions de ces cumul moyens de récompenses sur 100 *runs* différents de l'algorithme. Autrement dit, nous présentons la distribution du cumul de récompenses sur les différents *runs* de l'algorithme comme une fonction du nombre de transitions observées. La barre sur la droite indique les pourcentages associés à l'histogramme.

La politique optimale a un cumul moyen de récompenses de 6 (calculé avec l'algorithme d'itération de la valeur sur un espace d'état-action très finement échantillonné). On peut voir sur la figure 1 que l'algorithme proposé permet d'apprendre des politiques quasi-optimales. Après 1000 transitions certaines politiques ont un score de 5 (85% de la politique optimale), ce qui est atteint par la majorité des politiques après 3000 transitions. Après 7000 transitions, des politiques très proches de l'optimale ont été trouvées pour quasiment tous les *runs* (le mode de la distribution associée est à 5.85, soit 98% de la politique optimale). Pour représenter la Q-fonction approximée, seulement 7.7 ± 0.7 fonctions noyaux ont été utilisées.

Deux remarques peuvent être faites sur ce test. Premièrement, le bruit d'observation est dépendant des entrées, étant donné qu'il modélise la stochasticité du PDM. Rappelons que nous avons choisi ici un bruit d'observation constant. Deuxièmement, le bruit peut ne pas être gaussien. Par exemple, à proximité de la chute d'eau, il aura tendance à être bimodal en raison de la discontinuité de la fonction de récompense. Rappelons que le filtre utilisé suppose que les bruits sont gaussiens. Ainsi l'algorithme proposé est relativement robuste, et il permet d'atteindre d'assez bons résultats si l'on considère que les observations étaient totalement aléatoires (aspect *off-policy*).

5.3 Problème du *mountain car*

Le second problème que nous testons est celui du *mountain car*. Un voiture sous-motorisée doit gravir une route à fort dénivelé. L'état est bi-dimensionel (position et vélocité) et continu, et il y a 3 actions possibles (avancer, reculer et neutre). La description du problème est donnée par Sutton & Barto (1998). Une récompense nulle est donnée à chaque pas de temps, et $r = 1$ est obtenu lorsque la voiture atteint le but.

Un premier problème est de trouver une paramétrisation pour cette tâche. Celle que nous avons discutée est adaptée aux problèmes continus, et non hybrides. Cependant elle peut aisément être étendue au problèmes à états continus et actions discrètes. Une solution simple consiste à avoir une paramétrisation pour chaque action discrète, c'est-à-dire de la forme $\theta = [\theta^{a_1}, \theta^{a_2}, \theta^{a_3}]$, la Q-fonction associée étant $Q_\theta(s, a) = Q_{\theta^a}(s)$. Toutefois, on peut noter que pour un état fixé et différentes actions les Q-valeurs vont être très proches, ou autrement dit $Q^*(s, a_1)$, $Q^*(s, a_2)$ et $Q^*(s, a_3)$ vont avoir des formes similaires, en tant que fonctions sur l'espace d'état. Nous considérons donc que les poids vont être spécifiques à chaque action, mais les centres et les largeurs des noyaux seront partagés. Plus formellement le vecteur de paramètres est

$$\theta = [(\alpha_i^{a_1})_{i=1}^p, (\alpha_i^{a_2})_{i=1}^p, (\alpha_i^{a_3})_{i=1}^p, (s_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p]^T$$

les notations étant les mêmes que dans les sections précédentes.

Comme pour le problème du *wet-chicken*, des transitions aléatoires ont été présentées à l'entrée du filtre. Les résultats sont présentés figure 2, qui est un histogramme bidimensionel similaire à celui introduit précédemment. La légère différence est que la mesure de performance est maintenant le *cost-to-go*, c'est-à-dire le nombre de pas de temps nécessaires pour atteindre le but. Ce coût peut être directement lié au cumul moyen de récompenses, nous l'utilisons car nous pensons que c'est une mesure plus significative pour ce problème. Pour chaque *run* de l'algorithme et toutes les 250 transitions, le *cost-to-go* moyen pour la politique courante a été calculé comme étant une moyenne sur 1000 épisodes aléatoirement initialisés (la moyenne est donc faite uniquement sur les états de départ, les transitions étant déterministes). La durée de vie de l'agent a été limitée à 1000 interactions avec l'environnement. L'histogramme est calculé sur 100 *runs*.

La politique optimale a un *cost-to-go* moyen de 55 (calculé avec une itération de la valeur sur un espace d'état très finement échantillonné). Il apparaît sur la figure 2 que l'algorithme proposé permet de trouver des politiques quasi-optimales. Après 1500 transitions, la plupart des politiques on un *cost-to-go* de moins de 120. Après 6000 transitions, des politiques très proches de l'optimale ont été trouvées pour quasiment tous les *runs* de l'algorithme (le mode de la distribution associée est à 60). Pour représenter la Q-fonction approchée, seulement 7.5 ± 0.8 fonctions noyaux ont été utilisées.

Ce problème n'est pas stochastique, mais les récompenses informatives sont très rares (ce qui pourrait causer une convergence trop rapide du gain de Kalman), les transitions sont non-linéaires et les récompenses binaires. Malgré cela, le filtre proposé montre un bon comportement. Une fois encore, nous considérons que l'algorithme proposé atteint d'assez bons résultats étant donné la tâche à traiter.

5.4 Comparaison aux autres méthodes

Nous soutenons que la qualité des politiques apprises est comparable aux méthodes de l'état de l'art. Mesurer cette qualité dépend de la configuration du problème et des mesures de performance, cependant le filtrage bayésien de la récompense permet d'obtenir des politiques quasi-optimales. Le lecteur peut se référer à Schneegass *et al.* (2006) pour quelques éléments de comparaison.

Dans la plupart des approches, le système est contrôlé pendant l'apprentissage, ou pour les méthodes hors-ligne, les exemples observés sont générés par une politique sous-optimale. Dans les expérimentations proposées, des transitions totalement aléatoires sont observées, et la façon dont est abordé le problème est plus proche de la régression pure. Cependant, pour la tâche du *mountain car*, il est souvent rapporté la nécessité de quelques centaines d'épisodes pour apprendre un politique quasi-optimale (voir par exemple Sutton & Barto, 1998), et chaque épisode devrait contenir de quelques dizaines à plusieurs centaines d'interactions avec l'environnement (cela dépend de la qualité du contrôle courant). Dans l'approche proposée quelques milliers de transitions doivent être observées pour atteindre une politique quasi-optimale. C'est très grossièrement le même ordre de grandeur en terme de vitesse de convergence.

Pour l'instant, nous traitons les différentes tâches de contrôle comme des problèmes de régression, l'apprentissage se faisant à partir de transitions aléatoires. Notre approche est donc pour l'instant difficilement comparable à l'état de l'art, pour lequel l'apprentissage se fait à partir de trajectoires. Il est cependant prévu

de conduire des comparaisons plus poussées à d'autres approches quand un organe de contrôle sera ajouté au paradigme proposé.

6 Conclusion et perspectives

Nous avons introduit une formulation de l'apprentissage par renforcement sous forme de filtrage bayésien. En observant les récompenses (et les transitions associées), le filtre est capable d'inférer une politique quasi-optimale via la Q-fonction paramétrée. Une implémentation spécifique, basée sur un FKSP, une paramétrisation non-linéaire par noyaux et une méthode de dictionnaire permettant une représentation parcimonieuse de la fonction de valeur a été décrite. Elle a été testée sur deux problèmes, chacun exhibant des difficultés spécifiques pour l'algorithme. Notre filtre *off-policy* s'est avéré efficace sur ces tâches continues.

Cependant, ce papier n'a fait qu'effleurer certaines des possibilités du canevas proposé. L'approche par filtrage bayésien permet de dériver une information d'incertitude sur la Q-fonction estimée, elle pourrait être utilisée pour le dilemme entre exploration et exploitation, dans l'idée de Dearden *et al.* (1998) ou de Strehl *et al.* (2006). Cela pourrait permettre d'accélérer et d'améliorer l'apprentissage. De plus, nous pensons que l'observabilité partielle devrait assez naturellement être incluse dans ce cadre de travail, la Q-fonction pouvant être considérée ici comme une fonction sur des densités de probabilité.

Nous avons l'intention pour la suite de traiter les deux problèmes mentionnés précédemment, notre objectif principal étant de regrouper dans un seul modèle ce que nous considérons comme les trois principaux problèmes de l'apprentissage par renforcement, à savoir la généralisation, l'observabilité partielle et le compromis entre exploration et exploitation. Cependant il existe d'autres points d'intérêt. Un bruit artificiel d'évolution plus efficace pourrait améliorer la stabilité et les qualités de convergence du filtre. Un bruit adaptatif d'observation pourrait s'avérer nécessaire pour des tâches plus complexes, étant donné qu'il est formellement dépendant des entrées. De plus, il y a encore quelques soucis techniques, comme la recherche du maximum sur les actions. Enfin, une étude théorique peut s'avérer nécessaire, notamment en ce qui concerne la convergence, qui peut être principalement compromise par l'opérateur $\max$.

Références

BELLMAN R. (1957). *Dynamic Programming*. Dover Publications, sixth edition.

BERTSEKAS D. P. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific, 3rd edition.

CARREIRA-PERPINAN M. A. (2000). Mode-finding for mixtures of gaussian distributions. *IEEE Transactions on Pattern Analalysis and Machine Intelligence*, **22**(11), 1318–1323.

CHEN Z. (2003). *Bayesian Filtering : From Kalman Filters to Particle Filters, and Beyond*. Rapport interne, Adaptive Systems Lab, McMaster University.

DEARDEN R., FRIEDMAN N. & RUSSELL S. J. (1998). Bayesian Q-learning. In *AAAI/IAAI*, p. 761–768.

ENGEL Y. (2005). *Algorithms and Representations for Reinforcement Learning*. PhD thesis, Hebrew University.

PHUA C. W. & FITCH R. (2007). Tracking value function dynamics to improve reinforcement learning with piecewise linear function approximation. In *ICML 07*.

SCHNEEGASS D., UDLUFT S. & MARTINETZ T. (2006). Kernel rewards regression : an information efficient batch policy iteration approach. In *AIA'06 : Proceedings of the 24th IASTED international conference on Artificial intelligence and applications*, p. 428–433, Anaheim, CA, USA : ACTA Press.

STREHL A. L., LI L., WIEWIORA E., LANGFORD J. & LITTMAN M. L. (2006). Pac model-free reinforcement learning. In *23rd International Conference on Machine Learning (ICML 2006)*, p. 881–888, Pittsburgh, PA, USA.

SUTTON R. S. & BARTO A. G. (1998). *Reinforcement Learning : An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 3rd edition.

VAN DER MERWE R. (2004). *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*. PhD thesis, OGI School of Science & Engineering, Oregon Health & Science University, Portland, OR, USA.

VAPNIK V. N. (1998). *Statisical Learning Theory*. John Wiley & Sons, Inc.

Planification optimiste de systèmes déterministes

Jean-François Hren and Rémi Munos

Projet SequeL, INRIA Lille - Nord Europe
40 avenue Halley, 59650 Villeneuve d'Ascq, France
jean-francois.hren@inria.fr and remi.munos@inria.fr

Résumé : Si l'on possède un modèle d'un système déterministe contrôlé, alors depuis tout état, on peut construire l'ensemble des états atteignables depuis cet état et utiliser toute séquence d'actions. Cela forme un arbre dont la taille est exponentielle en l'horizon temporel de la planification. Nous nous posons alors la question : étant donné un nombre fini de ressources computationnelles, (e.g. temps processeur), qui peut ne pas être connu à l'avance, quelle est la meilleure façon d'explorer cette arbre, telle qu'une fois tout les ressources computationnelles épuisées, l'algorithme soit dans la capacité de nous fournir une action (ou une séquence d'actions) dont la performance est aussi proche que possible de l'optimale ? La performance en fonction de l'optimale est définie comme le regret résultant du choix de l'action générée par l'algorithme en lieu et place de l'action optimale. Dans cet article nous étudions une exploration optimiste de l'arbre, où les états les plus prometteurs sont explorés en premiers, et comparons cette approche avec une exploration naïve uniforme. Des bornes sur le regret sont définies pour chacune des stratégies d'exploration uniforme et optimiste. Des expérimentations numériques illustrent les bien faits de la plannification optimiste. **Mots-clés** : planification, prise de décision optimiste, bornes supérieures, échantillonnage optimal, bornes sur le regret

1 Introduction

Cet article s'intéresse au problème d'utilisation optimale des ressources numériques disponibles (en terme de temps processeur, de mémoire disponibled, du nombre d'appels à un modèle boite-noire, ...) dans le but de résoudre des problèmes de prises de décisions séquentielles. Nous voulons construire un algorithme *anytime*, i.e. qui retourne une meilleure solution dés que des ressources additionnelles sont fournies. Nous sommes intérressés dans la génération d'une politique proche-optimale pour un système déterministe avec des récompenses dépréciées, un espace d'actions fini mais un espace d'états grand (voir infini). Nous supposons avoir un modèle génératif de la dynamique des états et des récompenses.

L'approche consiste à considérer à chaque pas de temps t, l'arbre des possibilités (look-ahead tree), qui peut être construit à partir de notre modèle, d'états atteignables en utilisant toutes séquences d'actions, débutant par l'état x_t. Une recherche est alors effectuée dans cet arbre en utilisant une stratégie d'exploration spécifique. Nous nous posons la question du comment doit-on explorer cet arbre de tel façon qu'après avoir utilisé une quantité fini de ressources numériques (ici nous considérons le nombre de nœuds ouverts ou de transitions qui est directement en relation avec le temps processeur ou le nombre d'appels à notre modèle), nous sommes en position de prendre la meilleure décision possible au sujet de l'action (ou de la séquence d'actions) à prendre depuis l'état x_t. Cette action est alors appliquée au système réel et l'ensemble du processus est ensuite répété depuis l'état x_{t+1}. Cette idée est proche de l'algorithme de *sparse sampling* de Kearns *et al.* (2002), où un outil d'échantillionnage est utilisé pour explorer l'arbre des possibilités et générer une action proche-optimal avec une grande probabilitée. Leurs contexte est plus général que le notre puisqu'ils considèrent les processus de décision markovien généraux alors que nous nous restreignons à des dynamiques et des récompenses déterministes. Cependant, le but de notre étude est l'exploration de l'arbre de façon efficace alors qu'eux ne considérent qu'une exploration uniforme. Des travaux empiriques, comme Péret & Garcia (2004), suggèrent qu'une exploration non-uniforme pourrait grandement améliorer les performances de la politique résultante, étant donné une quantité fixée de ressources computationnelles.

Les bornes sur les performance que nous obtenons ici ne dépendent pas de la dimension de l'espace des états, contrairement aux approches tradionnelles que sont la programmation dynamique et l'apprentissage par renforcement (où une politique proche-optimal est déduite de l'approximation d'une fonction valeur sur l'ensemble du domaine), voir Puterman (1994); Bertsekas & Tsitsiklis (1996); Sutton & Barto (1998),

qui sont sujet à la malédiction de la dimension. Toutefois nos bornes sont liées au facteur de branchement des arbres associés et de l'horizon temporel $1/(\log 1/\gamma)$ (où γ est le facteur de dépréciation). La mesure de performance que nous considèrons ici est le regret $R(n)$, qui est la perte de performance (en fonction de l'optimal) de la décision retournée par l'algorithme après avoir exploré l'arbre en utilisant n unités de ressources computationnelles. Cette notion sera développée dans la prochaine section. Si nous considérons une exploration uniforme de l'arbre, on obtient les bornes (supérieure et inférieure) du regret : $R(n) = \Theta(n^{-\frac{\log 1/\gamma}{\log K}})$, où K est le nombre d'actions (le facteur de branchement de l'arbre complet des possibilités). Nous espérons alors qu'une telle approche est intéressante quand l'espace des états est grand mais que le nombre d'actions est relativement petit.

Dans cet article, nous étudions une **exploration optimiste** de l'arbre, où les nœuds les plus prometteurs sont explorés en premiers. Cette exploration fonctionne en ouvrant à chaque tour le nœud qui a une possibilité d'être le meilleur (ce qui motive l'utilisation du terme "optimiste"), dans le sens où il possède la borne supérieure la plus élevée. L'idée de selectionner les actions basées sur des bornes supérieures de confiance (UCB) remonte aux travaux sur les bandits à plusieurs bras, voir Robbins (1952); Lai & Robbins (1985), et plus récemment Auer *et al.* (2002). La planification dans un contexte d'incertitudes en utilisant UCB a été étudié dans Kocsis & Szepesvari (2006) et l'algorithme en résultant UCT (UCB applied to Trees) a été appliqué avec succés au problème de recherche arborescente à grande échelle du go, voir Gelly *et al.* (2006). Cependant, les analyses du regret montre que UCT peut fournir des performances assez pauvre dû aux assomptions excessivement optimiste des bornes, voir Coquelin & Munos (2007). Nos travaux sont proche dans l'esprit à l'algorithme BAST (Bandit Algorithm for Smooth Trees) de Coquelin & Munos (2007), où nous utilisons le caractère smooth inhérent à l'arbre des possibilités (qui vient du fait que nous considérons la somme des récompenses dépréciées le long des chemins) pour définir des vraies bornes supérieures sur la valeur des nœuds. Utilisant cette exploration optimiste, notre résultat principale est une borne supérieure sur le regret $R(n) = O(n^{-\frac{\log 1/\gamma}{\log \kappa}})$, où $\kappa \in (1, K]$ est le facteur de branchement d'un arbre lié, composé de tous les nœuds qui seront eventuellement évalués en vu de déterminer leur appartenance ou non à un chemin optimal. Il intéressant de remarquer que l'exploration optimiste n'est pas moins bonne que l'exploration uniforme (dans certains sens qui seront détaillés plus tard), mais lui est bien meilleur dès que κ est plus petit que K et proche de 1. Nous montrons que κ est lié à la proportion de chemin proche-optimaux dans l'arbre complet des possibilités, et que dans des cas difficiles de problèmes de recherche, κ est petit comparé à K, ce qui accroît le bénéfice de l'utilisation de l'exploration optimiste vis à vis de celle uniforme. De plus, nous montrons que dans certains cas non-triviaux $\kappa = 1$ et nous en dérivons des bornes exponentielles.

Dans la section suivante, nous introduisons les notations et les motivations au sujet du problème d'exploration d'arbre. Ensuite, nous étudions les stratégies d''exploration uniforme et optimiste. Nous concluons l'article par des experimentations numériques illustrant les apports de l'exploration optimiste.

2 Planification avec des ressources numériques finies

Nous nous intéressont à l'utilisation optimale des ressources numériques disponibles pour la prise de décision. Dans cette optique, nous supposons que nous avons accès à un modèle génératif qui peut être utilisé pour générer des transitions et des récompenses simulées. La procédure de selection d'action (appellons la $\mathcal{A}$) prend en entrée l'état courant du systéme et renvoie une action $\mathcal{A}(n)$ (ou une séquence d'action, mais nous allons nous concentrer sur le cas du renvoi d'une seule action) en utilisant un nombre fini n de ressources numériques disponibles. Le terme ressource désigne un temps de calcul qui peut est mesuré e.g. en terme de temps processeur or du nombre d'appels au modèle génératif. La quantité n de ressources disponibles peut ne pas être connue avant qu'elle soit totalement epuisée donc nous souhaitons construire un algorithme anytime. Notre but est que l'action choisie $\mathcal{A}(n)$ soit la plus proche possible de l'action optimale dans cet état, dans le sens où, le regret $R_{\mathcal{A}}(n)$ (sur la somme des récompenses dépréciées à venir) d'avoir choisie cette action en lieu et place de l'optimale doit être faible. Nous allons maintenant introduire les notations nous permettant de définir plus précisement la notion de regret.

Nous considérons ici un systéme déterministe contrôlé définit par le tuple (X, A, f, r), où X est l'espace d'état, A est l'espace d'action, $f : X \times A \to X$ la dynamique des transitions, et $r : X \times A \to \mathbb{R}$ la fonction récompense. Si à l'instant t, le systéme est dans l'état $x_t \in X$ et que l'action a_t est choisie, alors le système transite vers le nouvel état $x_{t+1} = f(x_t, a_t)$ et une récompense $r(x_t, a_t)$ est reçue. Nous supposons que l'espace d'état est grand (éventuellement infini), et l'espace d'action est fini, avec K actions possibles. Nous considérons un problème avec un horizon temporel infini avec des récompenses dépréciées

($0 \leq \gamma < 1$ est le facteur de dépréciation). Pour toute politique $\pi : X \to A$ nous définissons la fonction valeur $V^\pi : X \to \mathbb{R}$ associée à la politique :

$$V^\pi(x) \stackrel{\text{def}}{=} \sum_{t \geq 0} \gamma^t r(x_t, \pi(x_t)),$$

où x_t est l'état du système à l'instant t en partant de l'état x (i.e. $x_0 = x$) et suivant la politique π. Nous définissons aussi la fonction Q-valeur $Q^\pi : X \times A \to \mathbb{R}$ associée à une politique π, dans l'état-action (x, a), comme :

$$Q^\pi(x, a) \stackrel{\text{def}}{=} r(x, a) + \gamma V^\pi(f(x, a)).$$

Nous avons la propriété que $V^\pi(x) = Q^\pi(x, \pi(x))$. Maintenant la fonction valeur optimale (respectivement la fonction Q-valeur) est définie comme : $V^*(x) \stackrel{\text{def}}{=} \sup_\pi V^\pi(x)$ (respectivement $Q^*(x, a) \stackrel{\text{def}}{=} \sup_\pi Q^\pi(x, a)$). Nous avons $V^*(x) = \max_a Q^*(x, a)$, les équations de programmation dynamique :

$$V^*(x) = \max_{a \in A} \left[r(x, a) + \gamma V^*(f(x, a)) \right]$$

$$Q^*(x, a) = r(x, a) + \gamma \max_{b \in A} Q^*(f(x, a), b).$$

Maintenant, reprenons l'algorithme de sélection d'action $\mathcal{A}$. Au tour n, l'algorithme retourne une action $\mathcal{A}(n)$. Le regret engendré par la selection de cette action en lieu et place de l'optimale est :

$$R_{\mathcal{A}}(n) \stackrel{\text{def}}{=} \max_{a \in A} Q^*(x, a) - Q^*(x, \mathcal{A}(n)). \tag{1}$$

Écrivons $\pi_{\mathcal{A}}$ la politique qui assigne à chaque état l'action retournée par l'algorithme de sélection d'action $\mathcal{A}$. Le résultat suivant nous confirme le choix de la définition précédente pour le regret dans le sens où un algorithme avec un faible regret génère une politique proche-optimale.

Proposition 1
Considérons un algorithme de contrôle utilisant une procédure de sélection d'action avec un regret ϵ (i.e. pour chaque état x, la procédure de sélection d'action retourne une action a telle que $Q^*(x, a) \geq V^*(x) - \epsilon$). Alors la performance de la politique résultante $\pi_{\mathcal{A}}$ est $\frac{\epsilon}{1-\gamma}$-optimale, i.e. pour tout x,

$$V^*(x) - V^{\pi_{\mathcal{A}}}(x) \leq \frac{\epsilon}{1 - \gamma}$$

Preuve 1
Soit T et T^π des opérateurs définis sur des fonctions bornées sur X tel que : Pour tout $V : X \to \mathbb{R}$ borné, $TV(x) \stackrel{\text{def}}{=} \max_{a \in A} \left[r(x, a) + \gamma V(f(x, a)) \right]$, $T^\pi V(x) \stackrel{\text{def}}{=} r(x, \pi(x)) + \gamma V(f(x, \pi(x)))$.

Nous avons la propriété que T et T^π sont des opérateurs de contraction en norme infinie, et que V^* est un point fixe de T (i.e. $V^* = TV^*$) et V^π est un point fixe de T^π (i.e. $V^\pi = T^\pi V^\pi$), voir e.g. Puterman (1994).

En utilisant ces notations, la supposition que $\mathcal{A}$ est une procédure de sélection d'action avec un regret ϵ s'écrit pour tout x, $T^{\pi_{\mathcal{A}}} V^*(x) \geq TV^*(x) - \epsilon$. Donc nous avons : $||V^* - V^{\pi_{\mathcal{A}}}||_\infty \leq ||TV^* - T^{\pi_{\mathcal{A}}} V^*||_\infty + ||T^{\pi_{\mathcal{A}}} V^* - T^{\pi_{\mathcal{A}}} V^{\pi_{\mathcal{A}}}||_\infty \leq \epsilon + \gamma ||V^* - V^{\pi_{\mathcal{A}}}||_\infty$, dont nous déduisons la proposition. $\square$

Notre but est de définir de meilleurs algorithmes de selection d'action $\mathcal{A}$ telle que, si fournis avec n unités de ressources computationnelles, ils retourneraient une action avec un regret minimal $R_{\mathcal{A}}(n)$.

La section suivante décrit une méthode basée sur la construction d'un arbre des possibilités uniforme.

3 Planification uniforme

3.1 Recherche dans l'arbre des possibilités

Étant donné un état x, nous décrivons une méthode pour sélectionner une action presque optimale, basé sur la construction d'un arbre des possibilités uniforme (uniform look-ahead tree). Nous considérons l'arbre (infini) $\mathcal{T}$ composé de tout les états atteignables depuis x : la racine correspond à x et chaque nœud de

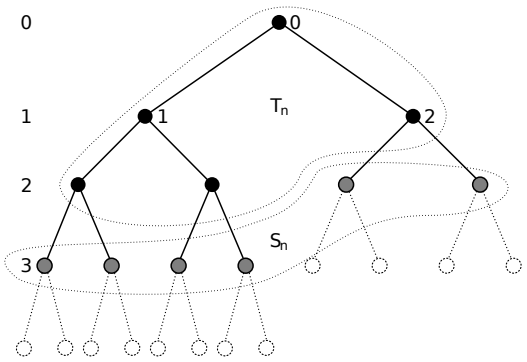


FIG. 1 – Ensemble des nœuds ouverts $\mathcal{T}_n$ (points noir) au tour $n = 5$ et ensemble des nœuds $\mathcal{S}_n$ (points gris) qui peuvent être ouverts ensuite. Ici $K = 2$.

profondeur d correspond à un état atteignable depuis x après une séquence de d transitions. Chaque nœud i (associé à un état y), a K fils $\mathcal{C}(i)$ (associés aux états $\{f(y,a)\}_{a\in A}$). Écrivons 0 le nœud racine, et $1 \dots K$ ses K fils (nœuds de profondeur 1).

Nous appelons un chemin dans $\mathcal{T}$ une séquence (finie ou infinie) de nœuds connectés débutant depuis la racine. Nous définissons la valeur v_i d'un nœud i comme le maximum, sur tout les chemins infinis passant par le nœud i, de la somme des récompenses dépréciées obtenues le long d'un tel chemin. Nous avons la propriété que $v_i = \max_{j\in\mathcal{C}(i)} v_j$, et que la valeur optimale (la valeur de la racine) $v^* = v_0 = \max_{i\in\mathcal{T}} v_i = \max\{v_1, \dots, v_K\}$.

Nous considérons que les ressources numériques sont exprimées en terme du nombre de nœuds ouverts. Cela est directement lié au temps processeur requis pour explorer la partie de l'arbre correspondante, ou la quantité de mémoire requis pour stocker l'information contenue dans les nœuds ouverts. Cela est aussi équivalent au nombre d'appels au modèle génératif, qui fournit l'état suivant $f(x,a)$ et la récompense $r(x,a)$.

Nous disons qu'un nœud est ouvert quand des ressources numériques sont allouées à ce nœud et à la génération des transitions vers ses fils (par un appel au modèle génératif pour chaque actions). Pour tout tour n, l'arbre ouvert $\mathcal{T}_n$ décrit l'ensemble des nœuds déjà ouverts. L'ensemble des nœuds dans $\mathcal{T}$ qui ne sont pas dans $\mathcal{T}_n$ mais dont le père est dans $\mathcal{T}_n$ est noté $\mathcal{S}_n$: cela représente l'ensemble des nœuds qui seront éventuellement ouverts dans un prochain tour (voir fig. 1).

Pour tout nœud $i \in \mathcal{S}_n$, nous définissons la valeur u_i comme la somme des récompenses dépréciées le long du chemin (fini) de la racine au nœud i (cette information est disponible au tour n car le père de i a été ouvert and donc la transition vers le nœud i a été générée). Maintenant, pour tout nœud $i \in \mathcal{T}_n$, nous définissons de façon récursive $u_i = \max_{j\in\mathcal{C}(i)} u_j$. Puisque ces u-valeurs ont été définies sur $\mathcal{S}_n$, elles sont aussi bien définies sur $\mathcal{T}_n$.

Notons que puisque les u-valeurs dépendent de $\mathcal{T}_n$, nous les écrirons $u_i(n)$ dès que leur dépendance en fonction de n est utile. À partir de leur définition, nous avons la propriété que $u_i(n)$ est une fonction croissante de n.

Nous supposons maintenant que toutes les récompenses sont dans l'intervalle $[0,1]$. Donc, puisque la somme des récompenses dépréciées depuis un nœud de profondeur d est tout au plus $\gamma^d + \gamma^{d+1} + \dots = \frac{\gamma^d}{1-\gamma}$, nous avons : pour tout $i \in \mathcal{T}_t \cup \mathcal{S}_t$ et $n \geq t \geq 1$, $u_i(n) \leq v_i \leq u_i(n) + \frac{\gamma^d}{1-\gamma}$.

3.2 L'algorithme

Ici nous considérons une politique d'exploration unifome, définie ainsi. Nous ouvrons en premier la racine. Ensuite, à chaque tour n, nous ouvrons un nœud dans $\mathcal{S}_n$ possédant la profondeur la plus faible. Voir Algorithme 1.

Ainsi, pour chaque tour, un arbre uniforme est ouvert : de plus, au tour $n = 1 + K + K^2 + \dots + K^d = \frac{K^{d+1}-1}{K-1}$ tout les nœuds de profondeur d ou moins ont été ouverts.

L'algorithme de sélection d'action uniforme retourne, après n tours, l'action qui correspond au fils de la racine $k \in \{1 \dots K\}$ qui a le plus grand u_k (les égalités sont départagées arbitrairement). En d'autres

Algorithme 1 Algorithme de planification uniforme $\mathcal{A}_U$

Set $n = 0$. Ouvrir la racine.
tantque Ressources numériques disponibles **faire**
  Ouvrir un nœud $i \in \mathcal{S}_n$ avec la plus faible profondeur.
  $n = n + 1$
fin tantque
return Action $\arg \max_{k \in \{1,\ldots,K\}} u_k(n)$

mots, après n tours, où nous avons ouvert l'arbre uniformément, $\mathcal{A}_U$ sélectionne l'action correspondant à la branche où nous avons trouvé le chemin (fini) avec la plus grande somme de récompenses.

3.3 Analyse

Théorème 1
Considérons l'algorithme de planification uniforme décrit précédemment. Ainsi pour toute fonction récompense, le regret de l'algorithme uniforme est borné par

$$R_{\mathcal{A}_U}(n) \leq \frac{1}{\gamma(1-\gamma)} \big[n(K-1)+1\big]^{-\frac{\log 1/\gamma}{\log K}}. \tag{2}$$

De plus, pour tout $n \geq 2$, il existe une fonction récompense, tel que le regret de cet algorithme sur ce problème est au moins

$$R_{\mathcal{A}_U}(n) \geq \frac{\gamma}{1-\gamma} \big[n(K-1)+1\big]^{-\frac{\log 1/\gamma}{\log K}}. \tag{3}$$

Nous déduisons la dépendance (dans le sens du pire cas) : $R_{\mathcal{A}_U}(n) = \Theta(n^{-\frac{\log 1/\gamma}{\log K}})$.

Preuve 2
Pour tout $n \geq 2$, soit d le plus grand entier tel que

$$n \geq \frac{K^{d+1}-1}{K-1}. \tag{4}$$

Donc tout les nœuds de profondeur d ont été ouverts. Donc pour tout $k \in \{1,\ldots,K\}$, nous avons $v_k \leq u_k + \frac{\gamma^{d+1}}{1-\gamma}$, car toute les récompenses jusqu'à la profondeur d ont été vues. Donc :

$$\begin{aligned} v^* &= \max_{k \in \{1,\ldots,K\}} v_k \leq \max_{k \in \{1,\ldots,K\}} u_k + \frac{\gamma^{d+1}}{1-\gamma} \\ &= u_{\mathcal{A}_U(n)} + \frac{\gamma^{d+1}}{1-\gamma} \leq v_{\mathcal{A}_U(n)} + \frac{\gamma^{d+1}}{1-\gamma}. \end{aligned}$$

Maintenant, à partir de (4), nous avons $d \geq \log_K[n(K-1)+1] - 2$, depuis laquelle nous déduisons que

$$v^* - v_{\mathcal{A}_U(n)} \leq \frac{\gamma^{d+1}}{1-\gamma} \leq \frac{1}{\gamma(1-\gamma)} \big[n(K-1)+1\big]^{-\frac{\log 1/\gamma}{\log K}}.$$

Pour la seconde partie du Théorème, étant donné n, définissons d comme le plus grand entier tel que (4) soit vrai. Donc, nous avons $d \leq \log_K[n(K-1)+1] - 1$. Définissons la fonction récompense suivante : tout les récompenses sont 0 excepté dans une branche (disons la branche 1) où les récompenses pour les transisitions des nœuds de profondeurs p à la profondeur $p+1$, où $p > d+1$, est 1. Donc $v^* = v_1 = \frac{\gamma^{d+2}}{1-\gamma}$ et $v_2 = 0$.

Donc à l'instant de la décision, tout les u_k (pour $k \in \{1,\ldots,K\}$) sont égales à zero (seul la récompense 0 a été observée), et une action arbitraire (disons l'action 2) est sélectionnée par l'algorithme. Donc $v^* - v_{\mathcal{A}_U(n)} = v_1 - v_2 = \frac{\gamma^{d+2}}{1-\gamma}$, et (3) en est déduit suivant la borne sur d. $\square$

Du Théorème 1 et de la Proposition 1, nous déduisons que pour garantir que les performances de la politique de planification uniforme $\pi_{\mathcal{A}(n)}$ soit ϵ-optimale (i.e. $||V^* - V^{\pi_{\mathcal{A}(n)}}||_\infty \leq \epsilon$), nous avons besoin d'allouer $n = \Theta\big(\frac{1}{\epsilon(1-\gamma)^2}\big)^{\frac{\log K}{\log 1/\gamma}}$ unités de ressources par étape de décision.

Dans cet article ; nous cherchons d'autres algorithmes de sélection d'action $\mathcal{A}$ qui peuvent faire une meilleure utilisation des ressources disponibles en minimisant le regret $R_{\mathcal{A}}(n)$ pour un montant fixé n de ressources computationnelles. Notons que l'analyse du pire cas considéré dans la seconde partie de la preuve du Théorème précédent pourrait nous décourager dans la recherche pour des meilleures bornes, puisqu'une analyse similaire peut être effectuée sur tout algorithme spécifique. Pourtant nous voudrions définir des classes de problèmes adaptées pour lesquelles nous pouvons espérer obtenir de meilleurs taux de convergence : nous ne pouvons espérer obtenir des meilleurs taux sur tout les problèmes mais ce doit être possible sur des classes de problèmes spécifiques.

4 Planification optimiste

Dans cette section, nous présentons un algorithme construisant un arbre des possibilités asymétrique visant à explorer en premier les parties de l'arbre les plus prometteuse.

4.1 L'algorithme

Nous définisson la b-valeur de tout nœud $i \in \mathcal{S}_n$ de profondeur d par $b_i \stackrel{\text{def}}{=} u_i + \frac{\gamma^d}{1-\gamma}$, et pour tout nœud $i \in \mathcal{T}_n$, sa b-valeur est définie récursivement par : $b_i \stackrel{\text{def}}{=} \max_{j \in \mathcal{C}(i)} b_j$.

Notons que, comme les u-valeurs, les b-valeurs sont aussi bien définies et qu'elles dépendent aussi de n, donc nous les écrirons $b_i(n)$ dès que la dépendance vis à vis de n est appropriée. Nous avons comme propriété que $b_i(n)$ est une fonction décroissante de n. Ainsi, puisque toutes les récompenses sont dans $[0, 1]$, nous avons directement la propriété que ces b-valeurs sont des bornes supérieures sur les valeurs des nœuds : pour tout $i \in \mathcal{T}_t \cup \mathcal{S}_t$, pour tout $n \geq t$, $u_i(n) \leq v_i \leq b_i(n)$.

La politique d'exploration optimiste consiste à ouvrir à chaque tour un nœud $i \in \mathcal{S}_n$ avec la plus grande b-valeur, voir Algorithme 2. L'action retournée est encore une fois le fils de la racine qui à la plus grande u-valeur (les égalités sont départagées arbitrairement).

Algorithme 2 Algorithme de planification optimiste $\mathcal{A}_O$

Set $n = 0$. Ouvrir la racine.
tantque Ressources numériques disponibles **faire**
  Ouvrir un nœud $i \in \mathcal{S}_n$ tel que $\forall j \in \mathcal{S}_n$, $b_i(n) \geq b_j(n)$.
  $n = n + 1$
fin tantque
return Action $\arg \max_{k \in \{1,\ldots,K\}} u_k(n)$

4.2 Analyse

Théorème 2
Considérons l'algorithme de planification optimiste décrit précédemment. Au tour n, le regret est borné par

$$R_{\mathcal{A}_O}(n) \leq \frac{\gamma^{d_n}}{1-\gamma}, \tag{5}$$

où d_n est la profondeur de l'arbre ouvert $\mathcal{T}_n$ (profondeur maximal des nœuds dans $\mathcal{T}_n$).

Une conséquence est que, pour toute fonction récompense, la borne supérieure sur le regret de la planification optimiste n'est jamais pire que celle de la planification uniforme (puisque l'exploration uniforme est une stratégie d'exploration qui minimise la profondeur d_n pour un n donné)

Preuve 3
Premiérement, remarquons que l'action retournée par l'algorithme optimiste correspond à la branche qui a été la plus profondément explorée. En effet, si ce n'était pas le cas, cela voudrait dire que si nous écrivions $i \in \mathcal{T}_n$ un nœud de profondeur maximale d_n, il existerait un nœud $j \in \mathcal{T}_n$ de profondeur $d < d_n$ tel que $u_j(n) \geq u_i(n)$. Donc il existerait un tour $t \leq n$ tel qu'un nœud i eût été ouvert au tour t, ce qui signifierait que $b_i(t) \geq b_j(t)$. Mais c'est impossible puisque $b_i(t) = u_i(t) + \frac{\gamma^{d_n}}{1-\gamma} \leq u_i(n) + \frac{\gamma^{d_n}}{1-\gamma} \leq u_j(n) + \frac{\gamma^{d_n}}{1-\gamma} < u_j(n) + \frac{\gamma^{d}}{1-\gamma} = b_j(n) \leq b_j(t)$.

Donc l'action retournée par l'algorithme correspond à la branche la plus profondément explorée. Soit $i \in \mathcal{T}_n$ un nœud de profondeur maximale d_n. i appartient à une des K branches reliées à la racine, disons la branche 1. Si l'action optimale est 1 alors la perte est de 0 et la borne supérieure est vérifiée. Autrement, l'action optimale n'est pas 1, disons que c'est 2. Soit $t \leq n$ le tour pendant lequel le nœud i a été ouvert. Cela signifie que $b_i(t) \geq b_j(t)$ pour tout les nœuds $j \in \mathcal{S}_t$, donc aussi pour tout les nœuds $j \in \mathcal{T}_t$. En particulier, $b_1(t) = b_i(t) \geq b_2(t)$. Mais $b_2(t) \geq b_2(n)$. Maintenant, les u-valeurs sont toujours des bornes inférieures sur les v-valeurs, donc $u_1(t) \leq v_1$, et de part la définition des u-valeurs, $u_i(t) \leq u_1(t)$. Nous avons donc :

$$\begin{aligned} v^* - v_{\mathcal{A}_O(n)} &= v_2 - v_1 \leq b_2(n) - v_1 \leq b_2(t) - v_1 \\ &\leq b_1(t) - u_1(t) \leq b_i(t) - u_i(t) = \frac{\gamma^{d_n}}{1-\gamma}. \end{aligned}$$

□

Remarque 1
Ce résultat montre que la borne supérieure pour la planification optimiste ne peut être pire que la borne supérieure pour la planification uniforme. Cela ne signifie pas que pour tout les problèmes, l'algorithme optimiste aura des performances au moins aussi bonne que celle de l'algorithme uniforme. En effet, si nous considérons l'exemple mentionné dans la preuve du Théorème 1, si à l'instant n toute les récompenses observées sont de 0, tout algorithme (comme l'optimiste) délivrerait une décision arbitraire, qui pourrait être pire que tout autre décision arbitaire (faite par l'algorithme uniforme par exemple). Cependant, si nous considérons des classes équivalentes de problèmes, où les classes sont définies par des arbres ayant la même fonction récompense à des permutations de branches près, alors nous affirmons que nous avons de meilleurs résultats tel que pour tout problème, la planification optimiste n'est jamais pire que la planification uniforme sur un problème d'une même classe. Cependant, nous n'approfondissons pas dans ce sens au sein de cet article.

Notons que la borne inférieure pour la planification uniforme s'applique aussi pour la planification optimiste (la preuve est la même : puisque jusqu'à un tour n toutes les récompenses sont de 0, la planification optimiste construira aussi un arbre uniforme). Cela montre que aucune amélioration ne peut être attendue, comme il le fût déjà mentionné. Pour quantifier les améliorations possibles vis à vis de la planification uniforme, des classes spécifiques de problèmes doivent être définies. Nous allons maintenant quantifier les classes de problèmes suivant la proportion de chemins proche-optimaux.

Pour tout $\epsilon \in [0, 1]$, définissons la proportion $p_d(\epsilon)$ de nœuds ϵ-optimaux de profondeur d :

$$p_d(\epsilon) \stackrel{\text{def}}{=} |\{\text{nœud } i \text{ de profondeur } d \text{ tel que } v_i \geq v^* - \epsilon\}| K^{-d},$$

et écrivons $p(\epsilon) \stackrel{\text{def}}{=} \lim_{d\to\infty} p_d(\epsilon)$ la proportion de chemins ϵ-optimaux dans l'arbre (notons que cette limite est bien définie car $p_d(\epsilon)$ est une fonction décroissante de d).

Théorème 3
Soit $\beta \in [0, \frac{\log K}{\log 1/\gamma}]$ tel que la proportion $p(\epsilon)$ de chemins ϵ-optimaux dans l'arbre $\mathcal{T}$ est $O(\epsilon^\beta)$. Définissons $\kappa \stackrel{\text{def}}{=} K\gamma^\beta \in [1, K]$.

Si $\beta < \frac{\log K}{\log 1/\gamma}$ (i.e. $\kappa > 1$), alors le regret de l'algorithme optimiste est

$$R_{\mathcal{A}_O}(n) = O\big(n^{-\frac{\log 1/\gamma}{\log \kappa}}\big).$$

Si $\beta = \frac{\log K}{\log 1/\gamma}$ (i.e. $\kappa = 1$), soit c une constante tel que $p(\epsilon) \leq c\epsilon^\beta$. Alors nous avons le taux exponentiel :

$$R_{\mathcal{A}_O}(n) = O\big(\gamma^{\frac{(1-\gamma)^\beta}{c'} n}\big),$$

où c' est une constante strictement supérieure à c.

Preuve 4
Soit le sous-arbre $\mathcal{T}_\infty \subset \mathcal{T}$ de tout les nœuds i de profondeur d qui sont $\frac{\gamma^d}{1-\gamma}$-optimaux, i.e.

$$\mathcal{T}_\infty \stackrel{\text{def}}{=} \bigcup_{d\geq 0} \big\{\text{nœud } i \text{ de profondeur } d \text{ tel que } v_i + \frac{\gamma^d}{1-\gamma} \geq v^*\big\}. \tag{6}$$

Prouvons que pour tout les nœuds ouverts par l'algorithme optimiste sont dans $\mathcal{T}_\infty$. En effet, soit i un nœud de profondeur d ouvert au tour n. Alors $b_i(n) \geq b_j$ pout tout $j \in \mathcal{S}_n \cup \mathcal{T}_n$, donc $b_i(n) = b_0(n)$ (b-valeur de la racine). Mais $b_0(n) \geq v_0 = v^*$, donc $v_i \geq u_i(n) = b_i(n) - \frac{\gamma^d}{1-\gamma} \geq v^* - \frac{\gamma^d}{1-\gamma}$, donc $i \in \mathcal{T}_\infty$.

Maintenant, de part la définition de β, il existe d_0 tel que la proportion de nœuds ϵ-optimaux de profondeur $d > d_0$ est tout au plus de $c\epsilon^\beta$, où c est une constante. Nous avons donc que le nombre n_d de nœuds de profondeur d dans $\mathcal{T}_\infty$ est borné par $c\big(\frac{\gamma^d}{1-\gamma}\big)^\beta K^d$.

Maintenant, écrivons d_n la profondeur de l'arbre ouvert $\mathcal{T}_n$ au tour n. Soit $n_0 = \frac{K^{d_0+1}-1}{K-1}$ le nombre de nœuds dans $\mathcal{T}$ de profondeur inférieure à d_0. Nous avons :

$$\begin{aligned} n \leq n_0 + \sum_{d=d_0+1}^{d_n} n_d &\leq n_0 + c \sum_{d=d_0+1}^{d_n} \big(\frac{\gamma^d}{1-\gamma}\big)^\beta K^d \\ &= n_0 + c' \sum_{d=d_0+1}^{d_n} \kappa^d \end{aligned}$$

avec $c' = c/(1-\gamma)^\beta$ et $\kappa = \gamma^\beta K$.

Premièrement, si $\kappa > 1$ alors nous avons $n \leq n_0 + c'\kappa^{d_0+1}\frac{\kappa^{d_n-d_0}-1}{\kappa-1}$. Donc $d_n \geq d_0 + \log_K \frac{(n-n_0)(\kappa-1)}{c'\kappa^{d_0+1}}$. Maintenant, grâce au Théorème 2, nous avons le regret :

$$\begin{aligned} R_{\mathcal{A}_O}(n) &\leq \frac{\gamma^{d_n}}{1-\gamma} = \frac{1}{1-\gamma}\left[\frac{(n-n_0)(\kappa-1)}{c'\kappa^{d_0+1}}\right]^{\frac{\log\gamma}{\log\kappa}} \\ &= O\big(n^{-\frac{\log 1/\gamma}{\log\kappa}}\big). \end{aligned}$$

Ensuite, si $\kappa = 1$ et si $p(\epsilon) \leq c\epsilon^\beta$ pour une constante c. Alors pour tout $c' > c$, il existe d_0 tel que pour tout $d \geq d_0$, $p_d(\epsilon) \leq c'\epsilon^\beta$. Soit $n_0 = \frac{K^{d_0+1}-1}{K-1}$. En suivant le même raisonnement que précédemment, nous déduisons que $n \leq n_0 + \frac{c'}{(1-\gamma)^\beta}(d_n - d_0)$, et le regret :

$$R_{\mathcal{A}_O}(n) \leq \frac{\gamma^{d_n}}{1-\gamma} = O\big(\gamma^{n\frac{(1-\gamma)^\beta}{c'}}\big).$$

□

Remarque 2
Remarquons que un β satisfaisant $p(\epsilon) = O(\epsilon^\beta)$ est inclut nécessairement dans l'interval $[0, \log_{1/\gamma} K]$. Les deux cas extrêmes sont

- *$\beta = 0$ signifie que tout les chemins sont optimaux, i.e. toute les récompenses sont égales. Cela correspond à $\kappa = K$.*
- *Si il y a un seul chemin où les récompenses sont de 1, et que toutes les autres récompenses sont de 0, alors, pour tout ϵ, la proportion de nœuds ϵ-optimaux de profondeur d est $1/K^d$ pour $d \leq d_0$ pour une profondeur d_0 pour laquelle $\frac{\gamma^{d_0}}{1-\gamma} \leq \epsilon$, et pour tout $d > d_0$ la proportion de nœuds ϵ-optimaux reste constante. Puisque $d_0 \geq \log_\gamma(1-\gamma)\epsilon$, la proportion de nœuds ϵ-optimaux de profondeur $d > d_0$ est tout au plus de $\big[(1-\gamma)\epsilon\big]^{\log_{1/\gamma} K}$, i.e. ce qui correspond à $\beta = \log_{1/\gamma} K$. C'est la plus grande valeur possible pour β. Cela correspond à $\kappa = 1$.*

De ce résultat, nous voyons que quand $\kappa > 1$, la borne pour l'algorithme optimiste est de l'ordre de $n^{-\frac{\log 1/\gamma}{\log\kappa}}$ tandis que pour l'uniforme, la borne est de l'ordre de $n^{-\frac{\log 1/\gamma}{\log K}}$. En regardant la preuve, nous

observons que κ joue le rôle du **facteur de branchement de l'arbre** $\mathcal{T}_\infty$ des nœuds ouvrables, de la même façon que K est le facteur de branchement de l'arbre complet $\mathcal{T}$. κ appartient à l'intervalle $[1, K]$, donc la borne supérieure sur le regret de la planification optimiste est toujours au moins aussi bon que celui pour l'uniforme. Il est meilleur que l'uniforme dès que $\kappa < K$, et vraiment meilleur quand κ est proche de 1. Notons que l'arbre $\mathcal{T}_\infty$ représente l'ensemble des nœuds i tel que étant donné les récompenses observées de la racine au nœud i, la presence de i dans un chemin optimal ne peut être affirmée. Cela représente l'ensemble des nœuds qui devraient être ouverts pour créer un chemin optimal. Ainsi la performance de l'algorithme optimiste est exprimée en fonction du facteur de branchement du sous-arbre $\mathcal{T}_\infty$ composé de tout les nœuds qui seront éventuellement ouverts. C'est un résultat d'optimalité fort qui nous permet d'en dériver des bornes inférieures en fonction de β (Théorème 4 ci-dessous).

Remarque 3
Le cas $\kappa = 1$ est aussi intéressant car il fournit des taux exponentiels plutôt que des taux polynomiaux. Ce cas est vérifié par exemple si il existe une profondeur d_0 tel que pour tout nœud i_d de profondeur $d \geq d_0$ le long d'un chemin optimal, si nous écrivons x_d l'état correspondant à ce nœud i_d, nous exigeons que la différence entre la fonction valeur optimale de x_d et la Q-valeur de toutes les actions sous-optimales soit plus grande qu'une constante $\Delta > 0$, i.e., $\exists \Delta > 0$, pour tout $d \geq d_0$,

$$V^*(x_d) - \max_{a \text{ t.q. } Q^*(x_d,a) < V^*(x_d)} Q^*(x_d, a) \geq \Delta. \tag{7}$$

En effet, si c'etait vrai, nous aurions pour tout $d \geq d_0$, pour tout fils non-optimal j de i_d, $v_j \leq v^* - \Delta\gamma^d$ (car $v_{i_d} = v^*$ puisque les nœuds i_d sont optimaux). Maintenant, le nombre de nœuds dans la branche j qui appartiennent à l'arbre ouvrable $\mathcal{T}_\infty$ est borné par K^{h-d-1} où h est la profondeur maximale tel que $v_j + \frac{\gamma^h}{1-\gamma} \geq v^*$, i.e., $\gamma^{h-d} \geq \Delta(1-\gamma)$. Donc $K^{h-d-1} = \frac{1}{K}\left[(1-\gamma)\Delta\right]^{-\beta}$ avec $\beta = \log K / \log(1/\gamma)$.

Donc le nombre de nœuds de profondeur $d \geq d_0$ dans $\mathcal{T}_\infty$ est borné par une constante indépendante de d, i.e. $\frac{1}{K}[\Delta(1-\gamma)]^{-\beta}$. Donc $p_d(\frac{\gamma^d}{1-\gamma}) \leq \frac{1}{K}\left[\Delta(1-\gamma)\right]^{-\beta}/K^d$. Donc $p_n(\epsilon) \leq \frac{1}{K}\left[\epsilon/\Delta\right]^\beta$. Donc $p(\epsilon) \leq \frac{1}{K}\left[\epsilon/\Delta\right]^\beta$ et nous avons $\kappa = 1$ et la constante $c = \frac{1}{K}(1/\Delta)^\beta$.

Remarque 4
Une extension naturelle du précédent cas (pour lequel nous déduisons $\kappa = 1$) est quand, depuis chaque état x_d correspondant à un nœud de profondeur $d \geq d_0$, pour une profondeur d_0, il existe un petit nombre m de séquences Δ-optimales de h actions, où $\Delta > 0$ est une constante donnée et h un entier donné. Cela signifie, que depuis x_d, il existe au plus m séquences d'actions $a_1, \ldots, a_h$, conduisant aux états $(x_{d+i})_{1 \leq i \leq h}$, tel que

$$\sum_{i=0}^{h-1} \gamma^i r(x_{d+i}, a_{i+1}) + \gamma^h V^*(x_{d+h}) \geq V^*(x_d) - \Delta.$$

Nous pouvons alors prouver que le facteur de branchement κ de l'arbre ouvrable est au plus de $m^{1/h}$. Remarquons que dans le cas présenté dans la remarque précédente nous avions $m = 1$ et $h = 1$ (et nous deduisions que $\kappa = 1$). Cependant, dans la remarque précédente, nous avons seulement supposé que la propriété (7) était vérifiée le long du chemin optimal (tandis que nous imposons ici qu'elle soit vérifié pour tout les nœuds). La preuve de ce résultat est plutôt technique et n'est pas incluse ici.

Nous donnons maintenant sans preuve[1] la borne inférieure pour la planification uniforme et optimiste dans cette classe spécifique de problèmes.

Théorème 4
Pour tout $\kappa \in [1, K]$ définissons $\mathcal{P}_\kappa$ comme la classe de problèmes tel que la proportion de chemins ϵ-optimaux est $O(\epsilon^\beta)$, où β est tel que $\kappa = K\gamma^\beta$ (comme définie dans le Théorème 3). Pour tout $\kappa \in [1, K]$,
- *il existe un problème dans $\mathcal{P}_\kappa$, tel que le regret de la planification optimiste sur ce problème est $\Omega(n^{-\frac{\log 1/\gamma}{\log \kappa}})$,*
- *il existe un problème dans $\mathcal{P}_\kappa$, tel que le regret de la planification uniforme sur ce problème est au moins $\Omega(n^{-\frac{\log 1/\gamma}{\log K}})$.*

[1] Un rapport de recherche avec la preuve sera disponible rapidement at *http ://sequel.futurs.inria.fr/hren/optimistic/*

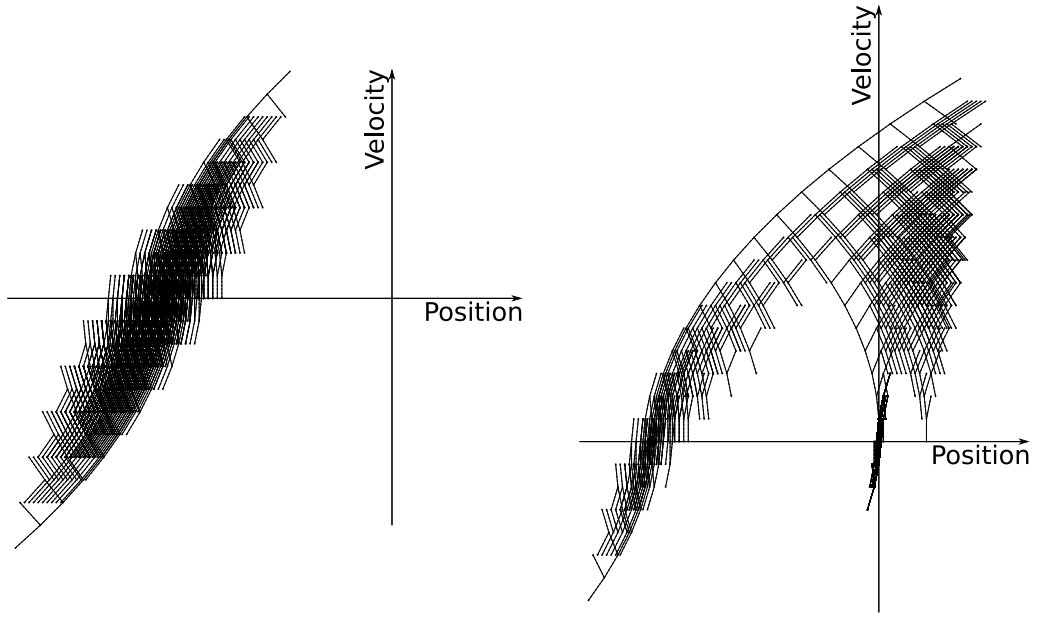


FIG. 2 – Arbres ouverts avec $n = 3000$ en utilisant les algorithmes uniforme (figure de gauche) et optimiste (figure de droite) en partant de l'état représenté par un point. La profondeur des arbres est de 11 pour l'uniforme et de 49 pour l'optimiste.

5 Expérimentations numériques

Nous avons fait des expérimentations numériques pour comparer les algorithmes de planification uniforme et optimiste. Nous considérons le système $\ddot{y}_t = a_t$, où un point défini par sa position y_t et sa vitesse v_t est contrôlé par une force (action) $a_t \in \{-1, 1\}$. Les dynamiques sont : $(y_{t+1}, v_{t+1})' = (y_t, v_t)' + (v_t, a_t)'\Delta t$, où $\Delta t = 0.1$ est le pas de temps discrétisé. La récompense de l'état (y_t, v_t) action a_t est définie par $\max(1 - y_{t+1}^2, 0)$ où y_{t+1} est la position de l'état suivant résultant.

Dans la figure 5 nous montrons les arbres résultants des algorithmes uniforme et optimiste utilisant le même nombre de ressources numériques $n = 3000$. L'état initial est $(y_0 = -1, v_0 = 0)$ et le facteur de dépréciation est $\gamma = 0.9$. L'arbre optimiste n'est pas profondément exploré sur la partie gauche de l'état initial car cela correspond aux états possédant de faibles récompenses, cependant, il est beaucoup plus profondément étendu le long de certaines branches fournissant de fortes récompenses, et en particulier, les branches menant les états à l'origne sont beacoup plus profondément étendu (profondeur maximale de 49).

Nous avons calculé le regret moyen des deux algorithmes pour $n = 2^{d+1} - 1$ avec $d \in \{2, \ldots, 18\}$ (arbres complet de profondeur d), où la moyenne est faite sur 1000 arbres où l'état initial a été échantillonné uniformément sur le domaine $[-1, 1] \times [-2, 2]$. La figure 3 montre le regret (en échelle logarithmique) pour les deux algorithmes. La pente $\frac{\log R(n)}{\log n}$ de ces courbes indique l'exposant dans la dépendance polynomiale du regret. Pour la courbe uniforme, nous calculons une pente numérique d'environ -1, donc le regret $R_{\mathcal{A}_U}(n) \simeq 1/n$. Pour la courbe optimiste, la pente numérique est envrion de $-3 = -\frac{\log 1/\gamma}{\kappa}$ laquelle correspond au facteur de branchement $\kappa = 1.04$. Le regret de la planification optimiste est donc de l'ordre de $1/n^3$ lequel est significativement un meilleur taux que l'uniforme, et explique les grandes améliorations de la planication optimiste vis à vis de l'uniforme. Par exemple, pour obtenir un regret de 0.0001 avec la planification uniforme, il faut plus de $n = 262142$ nœuds ouverts tandis que la planification optimiste n'en requière que $n = 4094$.

Nous n'avons pas la place de décrire d'autres applications plus ambitieuses mais d'autres simulations, incluant le double pendule inversé reliés par un resort, sont décrites à l'adresse : *http ://sequel.futurs.inria.fr/hren/optimistic/*.

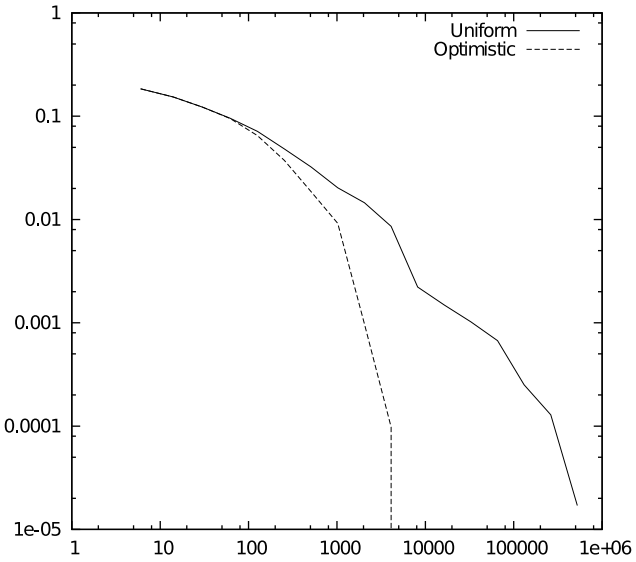


FIG. 3 – Regret moyen $R(n)$ pour les deux algorithmes comme une fonction de $n = 2^{d+1} - 1$ avec $d \in \{2, \ldots, 18\}$, en échelle logarithmique. La pente de chaque courbe indique l'exposant dans la réel dépendance polynomiale du regret.

6 Conclusions and travaux futurs

Une première extension de ce travail serait de considérer le cas des récompenses stochatiques, comme dans Coquelin & Munos (2007), où un terme additionnel (venant d'une borne de Chernoff-Hoeffing) est ajouté à la b-valeur pour définir des bornes supérieures de confiance à forte probabilité. Une seconde extensions, plus ambitieuse, serait de considérer le cas des transitions stochastiques. La possibilité de combiner cette technique de recherche pure avec des approximations locales de fonction valeur optimale vaut surement la peine d'être considéré.

Références

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

BERTSEKAS D. P. & TSITSIKLIS J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

COQUELIN P.-A. & MUNOS R. (2007). Bandit algorithms for tree search. *Uncertainty in Artificial Intelligence*.

GELLY S., WANG Y., MUNOS R. & TEYTAUD O. (2006). Modification of UCT with patterns in Monte-Carlo go. *Technical Report INRIA RR-6062*.

KEARNS M., MANSOUR Y. & NG A. (2002). A sparse sampling algorithm for near-optimal planning in large Markovian decision processes. In *Machine Learning*, volume 49, p. 193–208.

KOCSIS L. & SZEPESVARI C. (2006). Bandit based monte-carlo planning. *European Conference on Machine Learning*, p. 282–293.

LAI T. L. & ROBBINS H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, **6**, 4–22.

PÉRET L. & GARCIA F. (2004). On-line search for solving large Markov decision processes. In *Proceedings of the 16th European Conference on Artificial Intelligence*.

PUTERMAN M. L. (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley and Sons.

ROBBINS H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematics Society*, **58**, 527–535.

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

Planification distribuée par fusions incrémentales de graphes

Damien Pellier[1], Ilias Belaidi[1]

Université Paris Descartes
Centre de Recherche en Informatique (Équipe IAD)
45, rue des Saints Pères
F-75270 Paris cedex 06
Damien.Pellier@univ-paris5.fr
Ilias.Belaidi@univ-paris5.fr

Résumé : Dans cet article, nous proposons un modèle générique et original pour la synthèse distribuée de plans par un groupe d'agents, appelé « planification distribuée par fusions incrémentales de graphes ». Ce modèle unifie de manière élégante les différentes phases de la planification distribuée au sein d'un même processus. Le modèle s'appuie sur les graphes de planification, utilisé en planification mono-agent, pour permettre aux agents de raisonner et sur une technique de satisfaction de contraintes pour l'extraction et la coordination des plans individuels. L'idée forte du modèle consiste à intégrer au plus tôt, *i.e.*, au sein du processus local de planification, la phase de coordination. L'unification de ces phases permet ainsi aux agents de limiter les interactions négatives entre leurs plans individuels, mais aussi, de prendre en compte leurs interactions positives, *i.e.*, d'aide ou d'assistance, lors de l'extraction de leurs plans individuels.

1 Introduction

La planification multi-agent est une problématique centrale de l'intelligence artificielle distribuée. Les applications qui peuvent bénéficier directement des avancées de ce domaine de recherche sont nombreuses, *e.g.*, la robotique Alami *et al.* (1998), les services web Wu *et al.* (2003), dans lesquelles les approches centralisées sont difficilement envisageables. Classiquement, la planification multi-agent se définit comme un processus distribué qui prend en entrée : une modélisation des actions réalisables par des agents ; une description de l'état du monde connu des agents et un objectif. Le processus doit alors retourner un ensemble organisé d'actions, *i.e.*, un plan, qui s'il est exécuté, permet d'atteindre l'objectif. Une telle définition permet facilement d'appréhender ce qu'est la planification, mais cache la complexité et le découpage des différentes phases du processus. La littérature distingue cinq phases :

1. *Une phase de décomposition.* Les agents raffinent l'objectif initial en sous objectifs atteignables individuellement par les agents. Les approches proposées reposent le plus souvent sur des techniques centralisées de planification, *e.g.*, HTN Younes & Simmons (2003) ou sur les graphes de planification Iwen & Mali (2002).
2. *Une phase de délégation.* Les agents s'assignent mutuellement les objectifs obtenus lors de la première phase. Cette étape s'appuie sur des techniques d'allocation de ressources dans un contexte centralisé Bar-Noy *et al.* (2001) ou dans un contexte distribué basées par exemple sur la négociation Zlotkin & Rosenschein (1990), l'argumentation Tambe & Jung (1999), ou encore la synchronisation Clement & Barrett (2003).
3. *Une phase de planification locale.* Au cours de cette phase, chaque agent essaie de trouver un plan individuel pour atteindre l'objectif qui lui est assigné à l'étape 2. Les techniques utilisées reposent sur les algorithmes classiques de planification mono-agent, *e.g.*, Blum & Furst (1997); Nau *et al.* (2003); Penberthy & Weld (1992).
4. *Une phase de coordination.* Lors de cette phase chaque agent vérifie que le plan local élaboré à l'étape précédente n'entre pas en conflit avec les plans locaux des autres agents. Contrairement à la planification mono-agent, cette étape est nécessaire pour garantir l'intégrité fonctionnelle du système,

i.e., assurer que le but de chaque agent reste atteignable dans le contexte global. La problématique de la coordination a été tout particulièrement étudiée dans le domaine des systèmes multi-agents notamment en la considérant comme un processus de fusion de plans Tonino *et al.* (2002) ou de résolution de conflits Cox & Durfee (2005).

5. *Une phase d'exécution.* Finalement, le plan exempt de conflit est exécuté par les agents.

Bien que cette décomposition soit conceptuellement correcte, l'expérience montre que les cinq phases sont souvent entrelacées : (*i*) les phases 1 et 2 sont souvent fusionnées à cause du lien fort qu'il existe entre les capacités des agents et la décomposition de l'objectif ; (*ii*) la dernière phase d'exécution conduit presque toujours à une replanification Fox *et al.* (2006) due à un événement non prévu lors de l'élaboration du plan et nécessite un retour aux phases 1, 2 et 3 ; (*iii*) la phase de coordination peut être réalisée avant Shoham & Tennenholtz (1995), après Tonino *et al.* (2002) ou pendant Lesser *et al.* (2004) l'étape de planification locale. Cette remarque fait clairement apparaître un manque de modèles capables d'embrasser les différentes phases du processus de planification multi-agent.

Pour tenter de répondre en partie à cette critique, nous présentons dans cet article un modèle de planification multi-agent dans lequel l'élaboration d'un plan partagé est vu comme un processus collaboratif et itératif, intégrant les quatre premières étapes décrites précédemment. L'idée forte du modèle proposé consiste à amalgamer au plus tôt, *i.e.*, au sein du processus local de planification, la phase de coordination en s'appuyant sur une structure classique dans le domaine de planification, les graphes de planification Blum & Furst (1997). Cette représentation possède trois principaux avantages : elle est calculable en temps polynomiale ; elle permet d'utiliser les nombreux travaux portant sur la recherche d'un plan solution dans les graphes de planification ; finalement, elle est assez générique pour représenter une certaine forme de concurrence entre les actions. Ce dernier point est un atout majeur dans le cadre de la planification multi-agent.

L'intégration des phases de coordination et de planification locale, au sein d'un même processus que nous appelons *fusions incrémentales de graphes*, présente l'avantage de limiter les interactions négatives entre les plans individuels des agents, mais aussi et surtout, de prendre en compte les interactions positives, *i.e.*, d'aide ou d'assistance entre agents. Supposons un instant que le but d'un agent soit de prendre un objet dans une pièce dont la porte est fermée à clé. Même si l'agent possède un plan pour atteindre son but une fois la porte ouverte mais qu'il n'est pas capable d'ouvrir la porte, le processus de planification échoue. Supposons maintenant qu'un second agent soit capable d'ouvrir la porte, il existe alors un plan solution. Faut-il encore que le premier agent soit en mesure de considérer, au moment de la synthèse de son plan local, l'aide que peut lui apporter le second agent, et que ce dernier soit en mesure de proposer l'action requise.

Dans la suite de l'article, nous présentons tout d'abord les définitions préliminaires nécessaires à la formalisation de notre approche, puis nous détaillons les cinq phases de l'algorithme de planification distribuée par fusions incrémentales de graphes : la décomposition du but global en buts individuels ; l'expansion et la fusion des graphes de planification ; l'extraction d'un plan individuel et finalement la coordination des plans individuels solution.

2 Définitions préliminaires

La syntaxe et la sémantique utilisées dans notre approche s'appuient sur la logique du premier ordre, à savoir, les constantes, les prédicats et les variables. Le nombre des symboles de prédicats et de constantes est fini. Les propositions sont des tuples d'arguments, *i.e.*, des constantes ou des variables, négatives ou positives, *e.g.*, at(?x, paris), ou ¬at(?x,paris). Pour les prédicats et les constantes, nous utiliserons des chaînes de caractères alphanumériques comportant au moins deux caractères. Les variables seront définies par un seul caractère précédé par ?. Nous définissons la co-désignation comme la relation d'équivalence entre les constantes et les variables. Par extension, deux propositions se co-désignent si elles sont toutes deux négatives ou positives, si elles possèdent le même nombre d'arguments et si tous leurs arguments se co-désignent deux à deux, *e.g.*, at(?x, paris) et at(?y, paris) se co-désignent si $?x = ?y$.

Les opérateurs de planification sont définis comme des fonctions de transition au sens STRIPS. Nous nous plaçons ici dans le cadre de la planification classique : actions instantanées, statiques, déterministes et observabilité totale.

Définition 1 (Opérateur)
Un opérateur o *est un triplet de la forme* $(name(o), precond(o), effects(o))$ *ou* $name(o)$ *définit le nom de l'opérateur,* $precond(o)$ *l'ensemble des préconditions de* o *à satisfaire et* $effects(o)$ *l'ensemble de ses effets.*

Par la suite nous noterons respectivement $effects^+(o)$ *les effets positifs et* $effects^-(o)$ *les effets négatifs d'un opérateur* o.

Définition 2 (Action)
Une action *est une instance d'un opérateur. Si* a *est une action et* s_i *un état tel que* $precond(a) \subseteq s_i$ *alors l'action* a *est* applicable *dans* s_i *et le résultat de son application est un état :* $s_{i+1} = (s_i - effects^-(a)) \cup effects^+(a)$.

Définition 3 (Indépendance)
Deux actions a *et* b *sont* indépendantes[1] *ssi* $effects^-(a) \cap (precond(b) \cup effects^+(b)) = \emptyset$ *et* $effects^-(b) \cap (precond(a) \cup effects^+(a)) = \emptyset$. *Par extension un ensemble* A *d'actions est indépendant si toutes les paires d'actions de* A *sont indépendantes deux à deux.*

Définition 4 (Graphe de planification)
Un graphe de planification $\mathcal{G}$ *est un graphe orienté par niveaux*[2] *organisés en une séquence alternée de propositions et d'actions de la forme* $\langle P_0, A_0, P_1, \ldots, A_{i-1}, P_i \rangle$. P_0 *contient les propositions de l'état initial. Les niveaux* A_i *avec* $i > 0$ *sont composés de l'ensemble des actions applicables à partir des niveaux propositionnels* P_i *et les niveaux* P_{i+1} *sont constitués des propositions produites par les actions de* A_i. *Les arcs sont définis de la manière suivante : pour toutes les actions* $a \in A_i$, *un arc relie* a *avec toutes les propositions* $p \in P_i$ *qui représentent les préconditions de* a *; pour toutes les propositions* $p \in P_{i+1}$ *un arc relie* p *aux actions* $a \in A_i$ *qui produisent ou suppriment* p. *À chaque niveau le maintien des propositions du niveau* i *au niveau* $i+1$ *est assuré par des actions appelées* no-op *qui ont une unique proposition comme précondition et effet positif.*

Définition 5 (Exclusion mutuelle)
Deux actions a *et* b *sont* mutuellement exclusives *ssi* a *et* b *sont dépendantes ou si une précondition de* a *est mutuellement exclusive avec une précondition de* b. *Deux propositions* p *et* q *d'un niveau propositionnel* P_i *d'un graphe de planification sont mutuellement exclusives si toutes les actions de* A_i *avec* p *comme un effet positif sont mutuellement exclusives avec toutes les actions qui produisent* q *au même niveau, et s'il n'existe aucune action de* A_i *qui produit à la fois* p *et* q. *Par la suite, nous noterons l'ensemble des exclusions mutuelles d'un niveau* A_i *et* P_i *respectivement* μA_i *et* μP_i.

Définition 6 (Point fixe)
Un graphe de planification $\mathcal{G}$ *de niveau* i *a atteint son* point fixe *si* $\forall i, i > j$, *le niveau* i *de* $\mathcal{G}$ *est identique au niveau* j, *i.e.,* $P_i = P_j$, $\mu P_i = \mu P_j$, $A_i = A_j$ *et* $\mu A_i = \mu A_i$.

Définition 7 (Agent)
Un agent *est un tuple* $\alpha = (name(\alpha), operators(\alpha), belief(\alpha))$ *où* $name(\alpha)$ *définit le nom de l'agent,* $operators(\alpha)$ *l'ensemble des opérateurs que peut réaliser l'agent, et* $belief(\alpha)$ *les croyances de l'agent.*

De plus, nous introduisons $precond(\alpha)$ qui définit l'ensemble des propositions utilisées dans la description des préconditions des opérateurs de α, $effects^+(\alpha)$ et $effects^-(\alpha)$ qui représentent respectivement l'ensemble des propositions utilisées dans la description des effets négatifs et positifs des opérateurs de α. Nous supposons que ces trois ensembles constituent la partie publique d'un agent, *i.e.*, les agents peuvent accéder à ces informations à n'importe quel moment du processus de planification. En d'autres termes, en rendant publique ces informations, un agent publie les propriétés du monde qu'il est en mesure de modifier.

Définition 8 (Interaction négative)
Une action a *d'un graphe de planification d'un agent* α menace *l'activité d'un agent* β *ssi une proposition* $p \in effects^-(a)$ *co-désigne une proposition* q *telle que* $q \in precond(\beta)$ *ou* $q \in effects^+(\beta)$.

Définition 9 (Interaction positive)
Une action a *d'un graphe de planification d'un agent* α assiste *l'activité d'un agent* β *ssi une proposition* $p \in effects^+(a)$ *co-désigne une proposition* q *telle que* $q \in precond(\beta)$ *ou* $q \in effects^+(\beta)$.

[1] L'indépendance entre actions n'est pas une propriété spécifique à un problème de planification donné. Elle découle uniquement de la description des opérateurs des agents.

[2] Un graphe dont les sommets sont partitionnés en ensembles disjoints tels que les arcs connectent seulement les sommets des niveaux adjacents.

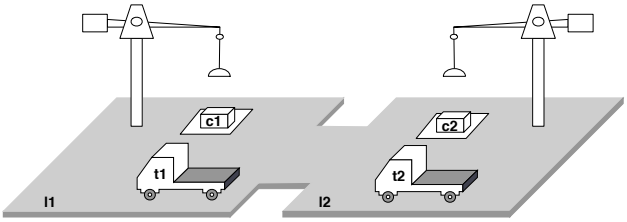


FIG. 1 – État initial de l'exemple :

$$\textit{belief}(\mathsf{ag1}) = \{\mathsf{at}(\mathsf{c1},\mathsf{l1}), \mathsf{at}(\mathsf{t1},\mathsf{l1})\}$$
$$\textit{belief}(\mathsf{ag2}) = \{\mathsf{at}(\mathsf{c2},\mathsf{l2}), \mathsf{at}(\mathsf{t2},\mathsf{l2})\}$$
$$\textit{belief}(\mathsf{ag3}) = \{\mathsf{at}(\mathsf{t1},\mathsf{l1}), \mathsf{at}(\mathsf{t2},\mathsf{l2})\}$$

Notons que ces deux définitions revêtent un caractère particulièrement important dans le cadre de notre modèle. C'est en effet, sur la base de ces interactions, que les agents seront capables de planifier en intégrant les activités des autres agents.

Définition 10 (Problème)
Un problème de planification multi-agent *est un tuple $\mathcal{P} = (\mathcal{A}, g)$ où $\mathcal{A}$ représente l'ensemble des agents devant résoudre le problème et g le but à atteindre, i.e., l'ensemble des propositions qui doivent être satisfaites par les agents après l'exécution du plan. Nous faisons l'hypothèse restrictive que l'union des croyances des agents d'un problème de planification est cohérente (cas classique de la planification mono-agent), i.e., pour deux agents α et $\beta \in \mathcal{A}$, si une proposition $p \in \textit{belief}(\alpha)$ alors $\neg p \notin \textit{belief}(\beta)$. Cependant, aucune hypothèse n'est faite sur le possible partage de croyances entre les agents en termes de faits ou d'opérateurs.*

Définition 11 (But individuel)
Le but individuel g_α d'un agent α pour un problème de planification multi-agent $\mathcal{P} = (\mathcal{A}, g)$ avec $\alpha \in \mathcal{A}$ est défini par $g_\alpha = \{p \in g \mid p$ co-designe une proposition $q \in \textit{effects}^+(\alpha)\}$. Autrement dit, un agent ne peut accepter comme but que les propositions qui sont définis comme des effets des actions qu'il peut exécuter.

Définition 12 (Plan)
Un plan *$\pi = \langle A_0, ..., A_n\rangle$ est une séquence finie d'ensembles d'actions avec $n \geq 0$. Si $n = 0$, alors le plan π est le plan vide, noté $\langle\rangle$. Si les ensembles d'actions composant le plan sont des singletons, i.e., $A_0 = \{a_0\}, \ldots, A_n = \{a_n\}$, nous utiliserons la notation simplifiée $\pi = \langle a_0, ..., a_n\rangle$.*

Définition 13 (Plan solution individuel)
Un plan $\pi_\alpha = \langle A_0^\alpha, ..., A_n^\alpha\rangle$ est un plan solution individuel *d'un problème de planification multi-agent $\mathcal{P} = (\mathcal{A}, g)$ pour un agent $\alpha \in \mathcal{A}$ ssi tous les ensembles d'actions A_i^α sont indépendants et l'application des A_i^α à partir de $\textit{belief}(\alpha)$ définit une séquence d'états $\langle s_0, \ldots, s_n\rangle$ telle que $g_\alpha \subseteq s_n$.*

Définition 14 (Plan solution global)
Un plan $\Pi = \langle A_0, ..., A_n\rangle$ est un plan solution global *d'un problème de planification multi-agent $\mathcal{P} = (\mathcal{A}, g)$ ssi il existe un sous-ensemble d'agents $\mathcal{A}'$ avec $\mathcal{A}' \subseteq \mathcal{A}$ tel que l'union des buts individuels g_α des agents $\alpha \in \mathcal{A}'$ est égale à g, que tous les agents $\alpha \in \mathcal{A}'$ possèdent un plan solution individuel $\pi_\alpha = \langle A_0^\alpha, ..., A_n^\alpha\rangle$ et que tous les ensembles d'actions $A_i = \bigcup A_i^\alpha$ sont indépendants.*

Exemple 2.1 (Dockers) Considérons un exemple simple avec trois agents : deux agents capables de charger et de décharger des conteneurs et, un troisième qui assure le déplacement d'un conteneur d'un endroit à un autre. Le premier docker ag1 est à l'emplacement l1. Le second docker est à l'emplacement l2. Le but du problème est $g = \{\mathsf{at}(\mathsf{c1},\mathsf{l2}), \mathsf{at}(\mathsf{c2},\mathsf{l1}), \mathsf{at}(\mathsf{t1},\mathsf{l2}), \mathsf{at}(\mathsf{t2},\mathsf{l1})\}$. Nous donnons ci-dessous la définition des opérateurs du problème ainsi que son état initial (cf. *fig.* 1) :

```
;; Le docker ag1 charge un conteneur ?c dans le camion ?t à l'emplacement l1
load(l1,?c,?t)
    precond:   at(?t,l1), at(?c,l1)
    effects:   ¬at(?c,l1), in(?t,?c)
```

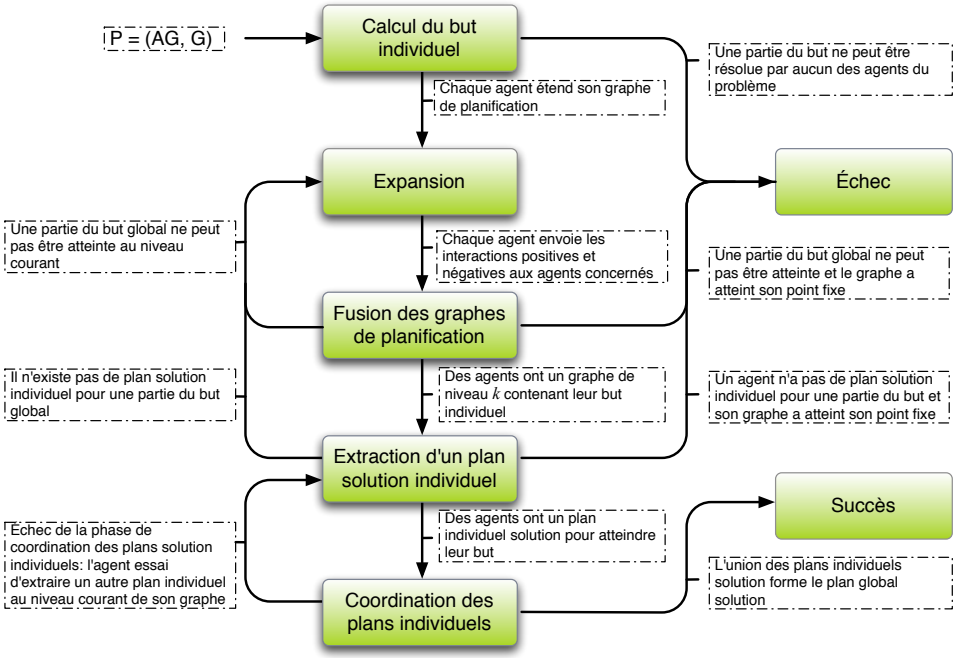


FIG. 2 – Organisation de la procédure de planification distribuée par fusions incrémentales de graphes.

;; Le docker ag1 décharge un conteneur ?c du camion ?t à l'emplacement l1
unload(l1,?c,?t)
precond: at(?t,l1), in(?t,?c)
effects: ¬in(?t,?c), at(?c,l1)

;; Le docker ag2 charge un conteneur ?c dans le camion ?t à l'emplacement l2
load(l2,?c,?t)
precond: at(?t,l2), at(?c,l2)
effects: ¬at(?c,l2), in(?t,?c)

;; Le docker ag2 décharge un conteneur ?c du camion ?t à l'emplacement l2
unload(l2,?c,?t)
precond: at(?t,l2), in(?t,?c)
effects: ¬in(?t,?c), at(?c,l2)

;; Le convoyeur déplace le camion ?t de l'emplacement ?from à l'emplacement ?to
move(?t,?from,?to)
precond: at(?t,?from)
effects: at(?t,?to), ¬at(?t,?from)

3 Dynamique du modèle

L'algorithme de planification distribuée par fusions incrémentales de graphes de planification (cf. *fig. 2*) s'articule autour de cinq phases. Il réalise une recherche distribuée en profondeur d'abord par niveau, découvrant une nouvelle partie de l'espace de recherche à chaque itération. Dans un premier temps, chaque agent calcule les buts individuels associés aux agents du problème puis entre dans une boucle dans laquelle il étend itérativement son graphe de planification, fusionne les interactions negatives et positives provenant de l'activité des autres agents et finalement, effectue une recherche locale d'un plan individuel solution. Si les agents réussissent à extraire un plan solution individuel à partir de leur graphe de planification, ils tentent alors de les coordonner. Dans le cas contraire, chaque agent étend une nouvelle fois son graphe de

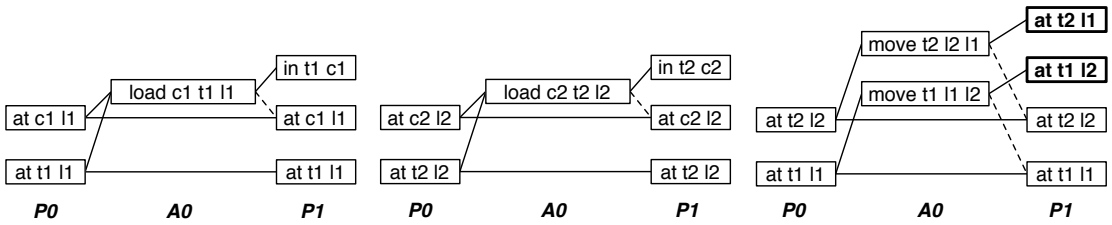
planification. La boucle d'expansion, de fusion, et de recherche de plans solution individuel se poursuit tant qu'aucun plan solution global n'est trouvé ou que la condition de terminaison sur échec n'est pas vérifiée.

3.1 Décomposition du but global

Tout d'abord, chaque agent calcule les buts individuels associés aux agents du problème (*def.* 11), *i.e.*, le sous-ensemble des propositions du but qui co-désignent un effet de ses opérateurs. Si une partie du but global n'apparaît dans aucun des buts individuels des agents, la phase de décomposition échoue, et la synthèse de plans se termine. Cet échec signifie qu'une partie du but global ne peut être atteinte, puisqu'aucun agent ne possède d'opérateurs capables de produire un sous-ensemble des propositions composant le but global. Dans le cas contraire, les agents passent dans la phase d'expansion. Considérons l'exemple 2.1, les buts individuels des agents sont définis de la manière suivante : $g_{\mathsf{ag1}} = \{\mathsf{at(c2,l1)}\}$; $g_{\mathsf{ag2}} = \{\mathsf{at(c1,l2)}\}$; $g_{\mathsf{ag3}} = \{\mathsf{at(t1,l2)}, \mathsf{at(t2,l1)}\}$. Il est important de noter qu'un agent peut ne pas avoir de but individuel, mais être nécessaire à la synthèse de plan. Dans ce cas, l'agent étend malgré tout son graphe pour lui permettre d'évaluer les actions d'aide qu'il peut proposer aux autres agents.

3.2 Expansion du graphe de planification

Cette phase débute par l'initialisation, par chaque agent, de son graphe de planification à partir de ses croyances. Puis, chaque agent étend son graphe en respectant la définition 4. Lorsque l'expansion de son graphe est terminée, l'agent envoie aux agents concernés les actions associées à son graphe qui peuvent interférer soit de manière positive (*def.* 9) soit de manière négative (*def.* 8). La figure 3 donne les graphes de planification associés aux agents $\mathsf{ag_1}$ (a), $\mathsf{ag_2}$ (b) and $\mathsf{ag_3}$ (c) après la premier expansion de leur graphe : les boites au niveau P_i représentent des propositions et les boites au niveau A_i des actions ; pour simplifier, les relations d'exclusion mutuelles ne sont pas représentées ; les lignes pleines modélisent les préconditions et les effets positifs des actions tandis que les lignes pointillées modélisent les effets négatifs ; finalement, les boites en gras définissent les propositions but lorsqu'elles sont atteintes sans mutex. Le graphe de $\mathsf{ag_3}$ contient son but individuel exempt d'exclusion mutuelle au niveau 1. En revanche, les graphes de planification de $\mathsf{ag_1}$ et $\mathsf{ag_2}$ ne contiennent par leur but individuel respectif. Considérons maintenant les interactions négatives et positives entre les actions des différents graphes. Par exemple, l'action move(t1,l1,l2) de $\mathsf{ag_3}$ au niveau 0 menace l'activité de $\mathsf{ag_1}$ car elle supprime la précondition at(t1,l1) qui peut être nécessaire à l'application de l'opérateur load et unload de $\mathsf{ag_1}$. Par opposition, cette même action de $\mathsf{ag_3}$ assiste l'activité de $\mathsf{ag_2}$ puisqu'elle produit l'effet at(t1,l2) qui peut être nécessaire au déclenchement des opérateurs load et unload de $\mathsf{ag_2}$. La liste complète des interactions entre les actions des agents est donnée par le tableau 1.



(a) Graphe de planification de l'agent 1 : but individuel non atteint.

(b) Graphe de planification de l'agent 2 : but individuel non atteint.

(c) Graphe de planification de l'agent 3 : but individuel atteint et exempt d'exclusion mutuelle.

FIG. 3 – Graphe de planification des agents de l'exemple 2.1.

Actions	*Agent*	*Négative*	*Positive*
load(c1,t1,l1)	$\mathsf{ag_1}$	–	$\mathsf{ag_2}$
load(c2,t2,l2)	$\mathsf{ag_2}$	–	$\mathsf{ag_1}$
move(t1,l1,l2)	$\mathsf{ag_3}$	$\mathsf{ag_1}$	$\mathsf{ag_2}$
move(t2,l2,l1)	$\mathsf{ag_3}$	$\mathsf{ag_2}$	$\mathsf{ag_1}$

TAB. 1 – Table des interactions entre agents au niveau 0.

3.3 Fusions des graphes de planification

Au cours de cette phase, l'agent fusionne les actions pouvant interférer avec son activité provenant des autres agents à son graphe de planification. Autrement dit, l'agent ajoute les actions à son graphe et achève son expansion *i.e.*, déclenche toutes actions possibles à partir des nouvelles propositions introduites au cours de l'expansion de son graphe. Le graphe contient désormais les actions provenant des autres agents pouvant interférer de manière positive ou négative avec sa propre activité. Deux cas sont à envisager :

1. Un agent α nécessaire à la réalisation du but global possède un graphe qui a atteint son point fixe au niveau i et $g_\alpha \not\subseteq P_i$ ou $g_\alpha \in P_i$ et $g_\alpha \cap \mu P_i \neq \emptyset$: la synthèse de plans échoue et l'algorithme se termine. En effet, un sous-ensemble des propositions du but global ne peut plus être atteint.
2. Un agent α nécessaire à la réalisation du but global possède un graphe qui n'a pas atteint son point fixe au niveau i mais $g_\alpha \not\subseteq P_i$ ou $g_\alpha \in P_i$ et $g_\alpha \cap \mu P_i \neq \emptyset$: tous les agents effectuent une nouvelle expansion de leur graphe de planification.

Finalement, si aucune des deux conditions n'est vérifiée, la phase de fusion se termine en garantissant qu'il existe un sous-ensemble des agents du problème capables d'atteindre le but global, et que chaque agent de ce sous-ensemble possède un graphe de planification de niveau i, tel que $g_\alpha \subseteq P_i$ et $g_\alpha \cup \mu P_i = \emptyset$.

À titre d'illustration, nous donnons à la figure 4 les graphes de planification des trois agents de l'exemple 2.1 après trois itérations d'expansion et de fusion. Afin de limiter l'ajout de propositions inutiles au niveau du graphe de planification pour ne pas pénaliser la recherche de plans, seules les propositions apparaissant dans la description des opérateurs d'un agent sont ajoutées. Par exemple, l'ajout de l'action move(t2,l2,l1) de l'agent ag3 dans le graphe de planification de l'agent ag1 ne produira qu'un seul effet, at(t2,l1), au niveau 1 du graphe de ag1. Le même principe s'applique également aux préconditions des actions ajoutées. Toutefois, si l'action n'est pas liée par au moins une précondition au niveau propositionnel (c'est le cas de l'action move(t2,l2,l1)), nous ajoutons une précondition fictive pour garantir l'existence d'un chemin de P_0 à P_i, et ainsi permettre l'extraction d'un plan individuel solution.

3.4 Extraction d'un plan individuel

L'extraction d'un plan solution est effectuée en s'appuyant sur une technique de satisfaction de contraintes proposée par Kambhampati (2000). Cette technique présente deux principaux avantages : d'une part, elle améliore de manière significative les temps d'extraction des plans solution et d'autre part, elle peut être facilement modifiée pour extraire un plan solution individuel intégrant les contraintes provenant des autres agents. L'extraction débute par l'encodage du graphe de planification sous la forme d'un CSP. Chaque proposition p à un niveau i du graphe de planification est assimilée à une variable CSP. L'ensemble des actions qui produisent p au niveau n constitue son domaine auquel on ajoute une valeur $\perp$ nécessaire à l'activation des actions dans le graphe. Les relations d'exclusion correspondent aux contraintes du CSP. Lorsque deux actions a_1 et a_2 sont mutuellement exclusives, alors pour toutes les paires de propositions p_1 et p_2 telles que a_1 peut produire p_1 et a_2 respectivement p_2, la contrainte $(p_1 = a_1) \Rightarrow (p_2 \neq a_2)$ est ajoutée au CSP. Lorsque deux propositions p_1 et p_2 sont mutuellement exclusives, on le traduit par une contrainte $\neg(p_1 \neq \perp) \wedge (p_2 \neq \perp)$. L'assignation des valeurs aux variables est dynamique car chaque assignation, à un niveau donné, active des variables au niveau précédent en respectant la procédure classique d'extraction de graphplan. Initialement, seules les variables représentant le but individuel de l'agent sont actives. L'activation dynamique se traduit par des contraintes d'activation au niveau du CSP. Autrement dit, lorsque une variable, représentant une proposition, prend une certaine valeur, *i.e.*, est produite par une action, alors d'autres variables, qui correspondent aux préconditions de l'action choisie, deviennent actives. L'extraction d'un plan solution individuel à partir du graphe de planification de l'agent est réalisée en résolvant le CSP. En partant des variables actives initialement, *i.e.*, le but individuel de l'agent, la procédure d'extraction cherche à assigner une valeur ou une action, à chaque variable ou proposition, pour satisfaire l'ensemble des contraintes, *i.e.*, des exclusions mutuelles. Si la résolution du CSP réussie, l'agent a extrait un plan solution individuel. Dans le cas contraire, l'agent n'est pas capable de produire un plan individuel solution. Deux cas doivent être considérés :

1. Un agent nécessaire à la réalisation du but global ne parvient pas à extraire un plan solution individuel même en poursuivant l'expansion de son graphe, l'algorithme se termine. Une partie du but global ne peut être démontrée.

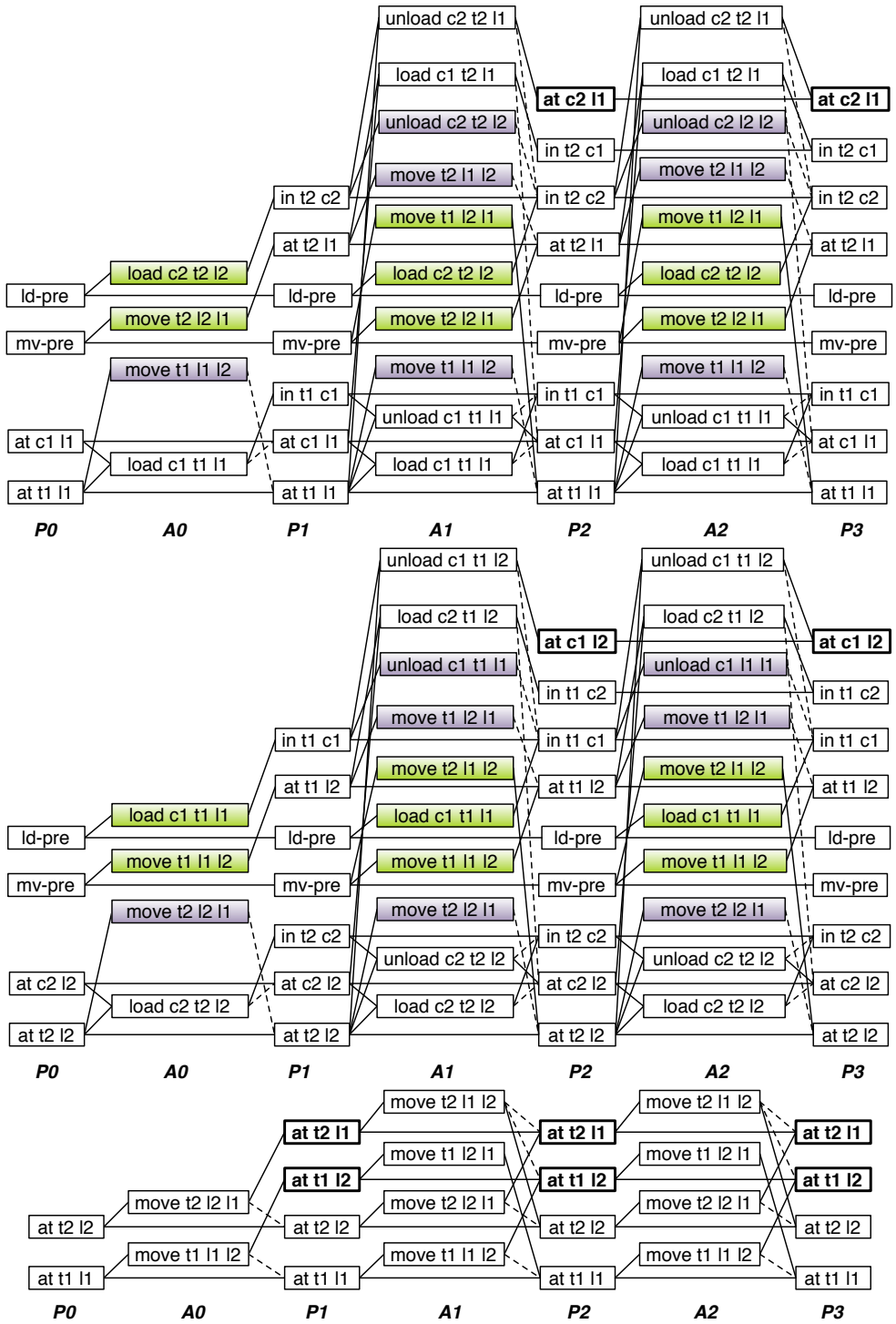


FIG. 4 – Graphes de planification des agents de l'exemple des dockers après la phase d'expansion et de fusion.

2. Un agent nécessaire à la réalisation du but global ne parvient pas à extraire un plan solution individuel mais peut espérer en trouver un en effectuant une nouvelle expansion de son graphe. Tous les agents effectuent une nouvelle expansion de leur graphe de planification.

Finalement, la phase d'extraction se termine en garantissant qu'il existe un sous-ensemble des agents du problème capables d'atteindre le but global, et possédant un plan individuel solution.

3.5 Coordination des plans individuels

Il s'agit maintenant de vérifier la compatibilité des plans individuels dans le contexte multi-agent. Rappelons que lors de la phase de fusion, les agents ont ajouté dans leur graphe les actions provenant d'autres agents et pouvant interférer avec leur propre activité. Par conséquent, un plan individuel solution peut contenir des actions qui doivent être exécutées par d'autres agents. Autrement dit, un plan solution individuel est un plan conditionnel *i.e.*, un plan exécutable si certaines contraintes sont vérifiées. Dans notre cas, ces contraintes sont définis par des couples (a, i) ou a représente une action et i le niveau où elle doit être exécutée. Considérons l'exemple des dockers et leur graphe de planification (cf. *fig.* 4). Les agents ag1 et ag2 peuvent extraire respectivement un plan solution individuel au niveau 2 :

$$\begin{aligned}\pi_{\mathsf{ag1}} &= \langle \mathsf{load}(\mathsf{c2},\mathsf{t2},\mathsf{l2}), \mathsf{move}(\mathsf{t2},\mathsf{l2},\mathsf{l1}), \mathsf{unload}(\mathsf{c2},\mathsf{t2},\mathsf{l1})\rangle \\ \pi_{\mathsf{ag2}} &= \langle \mathsf{load}(\mathsf{c1},\mathsf{t1},\mathsf{l1}), \mathsf{move}(\mathsf{t1},\mathsf{l1},\mathsf{l2}), \mathsf{unload}(\mathsf{c1},\mathsf{t1},\mathsf{l2})\rangle\end{aligned}$$

Le plan π_{ag1} est valide si le plan individuel de ag2 respecte la contrainte $(\mathsf{load}(\mathsf{c2},\mathsf{t2},\mathsf{l2}), 0)$ et celui de ag3 la contrainte $(\mathsf{move}(\mathsf{t2},\mathsf{l2},\mathsf{l1}), 1)$. De manière symétrique, π_{ag2} est valide si le plan individuel de ag2 respecte $(\mathsf{load}(\mathsf{c1},\mathsf{t1},\mathsf{l1}), 0)$ et celui de ag3 la contrainte $(\mathsf{move}(\mathsf{t1},\mathsf{l1},\mathsf{l2}), 1)$. De son côté, l'agent convoyeur ag3 peut potentiellement extraire plusieurs plans individuels solution dont aucun n'implique des actions provenant des autres agents :

$$\begin{aligned}\pi_{\mathsf{ag3}} &= \langle \{\mathsf{move}(\mathsf{t2},\mathsf{l2},\mathsf{l1}), \mathsf{move}(\mathsf{t1},\mathsf{l1},\mathsf{l2})\}, \mathsf{no\text{-}op}, \mathsf{no\text{-}op}\rangle \\ \pi'_{\mathsf{ag3}} &= \langle \mathsf{move}(\mathsf{t2},\mathsf{l2},\mathsf{l1}), \mathsf{move}(\mathsf{t1},\mathsf{l1},\mathsf{l2}), \mathsf{no\text{-}op}\rangle \\ \pi''_{\mathsf{ag3}} &= \ldots\end{aligned}$$

Les agents débutent la phase de coordination par l'échange des contraintes de leur plan individuel, et tentent ensuite d'intégrer les contraintes reçues à leur plan. L'intégration des contraintes repose sur le principe de moindre engagement. Tout d'abord, l'agent teste si les contraintes peuvent être ajoutées directement à son plan individuel solution. Ce teste ne nécessite aucune replanification, puisqu'il suffit de vérifier que l'action à ajouter n'est pas mutuellement exclusive avec une action déjà présente dans le graphe de planification de l'agent au même niveau. Dans notre exemple, ce mécanisme de coordination sera utilisé par les agents ag1 et ag2 pour prendre respectivement en compte les contraintes $(\mathsf{load}(\mathsf{c2},\mathsf{t2},\mathsf{l2}), 0)$ et $(\mathsf{load}(\mathsf{c1},\mathsf{t1},\mathsf{l1}), 0)$. Si ce premier mécanisme échoue, l'agent tente alors d'extraire un nouveau plan individuel solution en incluant les contraintes. En d'autres termes, il propage les contraintes dans le graphe de contraintes représentant son graphe de planification, et effectue l'extraction d'un nouveau plan individuel solution. Ce mécanisme sera utilisé par l'agent ag3 pour intégrer les contraintes $(\mathsf{move}(\mathsf{t1},\mathsf{l1},\mathsf{l2}), 1)$ et $(\mathsf{move}(\mathsf{t2},\mathsf{l2},\mathsf{l1}), 1)$ provenant des agents ag1 et ag2. Au final, les plans individuels solutions de l'exemple 2.1 obtenus après la phase de coordination sont les suivants :

$$\begin{aligned}\pi_{\mathsf{ag1}} &= \langle \mathsf{load}(\mathsf{c1},\mathsf{t1},\mathsf{l1}), \mathsf{no\text{-}op}, \mathsf{unload}(\mathsf{c2},\mathsf{t2},\mathsf{l1})\rangle \\ \pi_{\mathsf{ag2}} &= \langle \mathsf{load}(\mathsf{c2},\mathsf{t2},\mathsf{l2}), \mathsf{no\text{-}op}, \mathsf{unload}(\mathsf{c1},\mathsf{t1},\mathsf{l2})\rangle \\ \pi_{\mathsf{ag3}} &= \langle \mathsf{no\text{-}op}, \{\mathsf{move}(\mathsf{t1},\mathsf{l1},\mathsf{l2}), \mathsf{move}(\mathsf{t2},\mathsf{l2},\mathsf{l1})\}, \mathsf{no\text{-}op}\rangle\end{aligned}$$

Finalement, si ce second mécanisme de coordination échoue, les agents retournent dans la phase d'extraction de plans individuel. L'algorithme de fusions incrémentales de plans peut alors se terminer (si aucun autre plan individuel ne peut être extrait) ou encore nécessité une nouvelle expansion des graphes de planification des agents.

4 Conclusion

Dans cet article, nous avons proposé un modèle générique et original pour la synthèse distribuée de plans par un groupe d'agents, appelé *planification distribuée par fusions incrémentales de graphes*. Le modèle semble correct et complet, et unifie de manière élégante les différentes phases de la planification distribuée au sein d'un même processus. En outre, il permet aux agents de limiter les interactions négatives entre leur plan individuel, mais également de prendre en compte leurs interactions positives, *i.e.*, d'aide ou d'assistance, au plus tôt, *i.e.*, avant l'extraction des plans individuels. Notre approche est actuellement en cours d'implantation en s'appuyant sur les librairies PDDL4J[3] et Choco[4]. Chose intéressante, les premiers

[3] `http://sourceforge.net/projects/pdd4j/`

[4] `http://choco-solver.net/`

résultats expérimentaux semblent indiquer que la complexité n'est pas dépendante du nombre d'agents présents dans le problème, mais uniquement du couplage entre les activités des agents, *i.e.*, des interactions positives. Plusieurs pistes pour la poursuite de ce travail sont envisagées :

– la prise en compte de la robustesse des plans. Par robustesse, nous entendons la capacité d'un plan à tolérer plus ou moins des aléas d'exécution ou des croyances erronées sur l'état du monde. Supposons que nous soyons capables de construire un faisceau de plans solutions, *i.e.*, un plan contenant différentes alternatives. La question qui se pose alors est la suivante : comment choisir les plans individuels les plus robustes minimisant ainsi le risque d'échec à l'exécution ?
– l'extraction de plans temporels flottants, *i.e.*, de plans dans lesquels les actions sont représentées par des intervalles pouvant être décalés dans le temps. Bien que la causalité soit suffisante pour un grand nombre d'applications, la prise en compte des aspects temporels est parfois nécessaire. La notion de plans flottants apparaît particulièrement intéressante dans un contexte multi-agent, car ils possèdent une grande flexibilité qui peut être mise à profit au cours de la phase de coordination et d'exécution.

Références

ALAMI R., FLEURY S., HERRB M., INGRAND F. & ROBERT F. (1998). Multi robot cooperation in the MARTHA project. *IEEE Robotics and Automation Magazine*, **5**(1), 36–47.

BAR-NOY A., BAR-YEHUDA R., FREUND A., NAOR J. & SCHIEBER B. (2001). A unified approach to approximating resource allocation and scheduling. *Journal of Association for Computing Machinery*, **48**(5), 1069–1090.

BLUM A. & FURST M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, **90**(1-2), 281–300.

CLEMENT B. & BARRETT A. (2003). Continual coordination through shared activities. In *Proceedings of the International Conference on Autonomous Agent and Muti-Agent Systems*, p. 57–67.

COX J. & DURFEE E. H. (2005). An efficient algorithm for multiagent plan coordination. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, p. 828–835, New York, NY, USA : ACM Press.

FOX M., GEREVINI A., LONG D. & SERINA I. (2006). Plan stability : Replanning versus plan repair. In *Proceedings of the International Conference on Planning and Scheduling*, California, USA : AAAI Press.

IWEN M. & MALI A. D. (2002). Automatic problem decomposition for distributed planning. In *Proceedings of the International Conference on Artificial Intelligence*, p. 411–417.

KAMBHAMPATI S. (2000). Planning graph as a (dynamic) CSP : Exploiting EBL, DDB and other CSP search techniques in graphplan. *Journal of Artificial Intelligence Research*, **12**(1), 1–34.

LESSER V., DECKER K., WAGNER T., CARVER N., GARVEY A., HORLING B., NEIMAN D., PODOROZHNY R., NAGENDRAPRASAD M., RAJA A., VINCENT R., XUAN P. & ZHANG X. (2004). Evolution of the gpgp/taems domain-independent coordination framework. In *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, p. 87–143 : Kluwer Academic Publishers.

NAU D., AU T., ILGHAMI O., KUTER U., MURDOCK W., WU D. & YAMAN Y. (2003). Shop2 : An HTN planning system. *Journal of Artificial Intelligence Research*, **20**(1), 379–404.

PENBERTHY J. & WELD D. (1992). UCPO : A sound, complete, partial order planner for ADL. In C. R. B. NEBEL & W. SWARTOUT, Eds., *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, p. 103–114 : Morgan Kaufmann Publishers.

SHOHAM Y. & TENNENHOLTZ M. (1995). On social laws for artificial agents societies : off-line design. *Artificial Intelligence*, **73**(1–2), 231–252.

TAMBE M. & JUNG H. (1999). The benefits of arguing in a team. *Artificial Intelligence Magazine*, **20**(4), 85–92.

TONINO H., BOS A., DE WEERDT M. & WITTEVEEN C. (2002). Plan coordination by revision in collective agent-based systems. *Artificial Intelligence*, **142**(2), 121–145.

WU D., PARSIA B., SIRIN E J. & NAU D. (2003). Automating daml-s web services composition using shop2. In *Proceedings of International Semantic Web Conference*.

YOUNES H. & SIMMONS R. (2003). VHPOP : Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, **20**(1), 405–430.

ZLOTKIN G. & ROSENSCHEIN J. (1990). Negotiation and conflict resolution in non-cooperative domains. In *Proceedings of the American National Conference on Artificial Intelligence*, p. 100–105, Boston, Massachusetts.

Composition automatique et dynamique de Services Web utilisant des techniques de planification

Mohamad EL Falou[1], Maroua Bouzid[1], Thierry Vidal[2] ,Abdel-Illah Mouaddib[1]

[1] Université de Caen, Laboratoire GREYC, France
{ melfalou, bouzid, mouaddib }@info.unicaen.fr
[2] IRISA - INRIA Rennes, France
thierry.vidal@irisa.fr

Résumé :
L'aptitude à répondre automatiquement à une requête utilisateur exigeant la composition d'un ensemble de Services Web (WS) a reçu un intérêt croissant. La plupart des approches existantes pour la composition des WS considèrent des requêtes explicites, spécifiant le sous-ensemble des activités et les contraintes entre elles, pour trouver l'ensemble des WS optimisés accomplissant ces activités sous des contraintes prédéfinies. Dans ce travail, nous proposons une approche fondée sur la notion de requête implicite, définie par un état initial et un état but. L'approche utilise des techniques de planification de l'intelligence artificielle, en étendant la définition des WS ainsi que la spécification des requêtes pour permettre la création et la suppression d'objets.

Mots-clés : Web Services, Planification, Composition, Arbre de Recherche, Graphplan.

1 Introduction

Les Services Webs (nous utiliserons l'acronyme anglophone WS) sont des composants logiciels distribués, actifs et modulaires qui peuvent être annoncées, découverts et invoqués sur Internet en utilisant des protocoles standards (Gottscalk, 2000). Ils communiquent avec leurs clients et avec d'autres WS en envoyant des messages XML. Nous pouvons citer à titre d'exemples une compagnie de services de voyages ou un service d'achat de livres (tel amazon.com). Beaucoup de langages ont été mis au point par la communauté industrielle et académique pour spécifier les WS : WSDL (*Web Service Description Language*) (Chinnici *et al.*, 2004), DAML-S (*Web Service Description for the Semantic Web*) (Wu *et al.*, 2003), BPEL4WS *Business Process Execution Language for WS* (Andrews *et al.*, 2003).

Les techniques de planification de l'intelligence artificielle peuvent contribuer à la résolution du problème de composition. En effet, les services peuvent être modélisés comme des actions et la composition comme un plan de connections des WS. Le système construit le plan en explorant les services disponibles. Il développe les différents états intermédiaires à partir de l'état initial, en appliquant les services, jusqu'à atteindre l'état but. Il retourne soit un plan qui est un ensemble de services permettant d'atteindre le but, soit l'échec.

La principale contribution de ce papier est l'extension du modèle d'actions pour gérer la création ou la suppression d'objets comme des effets des actions. Cette contribution nous permet de répondre à de nouvelles requêtes plus expressives, appelées requêtes implicites, dans lesquelles les buts peuvent contenir des objets qui ont été générés par le plan.

2 Travaux dans le domaine

Les travaux sur la composition des WS à l'université de Trento présentés dans (Pistore *et al.*, 2004) décrivent les WS en tant que processus abstraits en utilisant le langage BPEL4WS pour la spécification et la composition des WS. Étant donnés n processus abstraits spécifiés en BPEL4WS, le système convertit automatiquement chacun d'entre eux en un système de transition d'états (STS). Un STS D est défini par $D = \langle S, A, I, T \rangle$ où S est l'ensemble d'états, A est l'ensemble des actions, $I \subset S$ est l'ensemble des états initiaux, et $T : S \times A \rightarrow 2^S$ est la fonction de transition. Après la conversion des WS, le système construit un produit parallèle $\sum_{\|}$ qui combine les n STS et permet aux n services d'évoluer ensemble.

À partir de ce produit parallèle $\sum_{\|}$, le système génère un domaine de planification et le fournit comme entrée au planificateur. Le problème de planification consiste à rechercher un plan π défini par un ensemble d'actions permettant d'atteindre un objectif tout en satisfaisant certaines contraintes. L'approche est fondée sur MBP (*Model Based Planner*) (Bertoli *et al.*, 2001) utilisant les techniques de model checking (Pistore & Traverso, 2001). L'inconvénient principal de cette approche consiste à recalculer $\sum_{\|}$ à chaque fois qu'on ajoute (ou on supprime) un service du domaine.

Dans sa thèse (Lazovik *et al.*, 2003), Alexander Lazovik définit le problème de la composition des WS de la même manière. Cependant, l'approche proposée permet au système d'interagir avec le client pendant la phase d'exécution. Cette interaction est fondée sur le principe de l'entrelacement des phases de planification et d'exécution. Le planificateur utilise les techniques de satisfaction de contraintes. L'auteur motive son travail par le fait que, dans un scénario de WS, les utilisateurs voudront savoir pourquoi certaines solutions sont préférables aux autres.

Une approche de composition de WS est proposée dans (McIlraith & Son, 2002) et se fonde sur le calcul de situations. Cette approche applique les techniques d'inférence logique sur un patron de plan prédéfini. Les services sont transformés sous forme d'actions exprimées dans GOLOG. Étant donnée une description du but, le langage de programmation logique GOLOG est utilisé pour instancier le plan approprié en composant les services. GOLOG est implémenté en Prolog et permet la spécification et l'exécution d'actions complexes dans les systèmes dynamiques.

Enfin, les auteurs de (Wu *et al.*, 2003) proposent une approche permettant d'exprimer une composition de tâches DAML-S en un problème de planification SHOP2(Nau *et al.*, 2003). SHOP2 est un planificateur utilisant un réseau hiérarchique de tâches (HTN) pour construire un plan à travers la décomposition de tâches.

Toutes les approches citées ci-dessus supposent que le domaine des objets est statique. En d'autres termes, il n'y a aucun moyen ni d'éliminer, ni de créer des objets. En outre, les requêtes pour la composition des WS sont définies de manière explicite. C'est cette limitation que nous souhaitons aborder.

Le reste de l'article est organisé comme suit : dans la section 3, nous présentons un exemple de problème de composition de WS. La section 4 est dédiée à la définition formelle de notre modèle. Puis dans la section 5, nous proposons deux algorithmes de planification basés respectivement sur *Tree-search* (Russel & Norvig, 2003) et *Graphplan* (Ghallab *et al.*, 2005). Enfin, nous concluons ce document par des résultats expérimentaux et la description des extensions à venir.

3 Exemple introductif

Considérons un ensemble de WS destinés à traiter des fichiers textes, des images et des fichiers audio :

1. **Traduction de textes**, WS_1 :
   ce WS dispose de trois services, $fr2en$ traduit un texte du français vers l'anglais, $en2ar$ traduit un texte de l'anglais vers l'arabe et $en2fr$ traduit un texte de l'anglais vers le français.
2. **Transformation des formats de fichier texte** WS_2 :
   ce WS dispose de deux services, $latex2doc$ transforme le format de fichier de $latex$ en doc et $doc2pdf$ transforme le format de fichier de doc en pdf.
3. **Fusion de fichiers textes**, WS_3 :
   ce WB dispose de deux services, $mergepdf$ fusionne deux fichiers pdf et $mergedoc$ fusionne deux fichiers doc.
4. **Transformation des images**, WS_4 :
   ce WS dispose de deux services, $png2gif$ transforme une image png en gif et $jpeg2png$ transforme une image $jpeg$ en png.
5. **Transformation de fichiers audios**, WS_5 :
   ce WS dispose de deux services, $wav2mp3$ transforme un fichier audio wav en $mp3$ et $rm2mp3$ transforme un fichier audio rm en $mp3$ également.

Supposons que nous ayons deux fichiers : le premier est en format doc rédigé en anglais, le second est en latex et en français. Nous voulons obtenir un fichier qui contient le contenu des deux textes traduits en arabe. Les approches existantes dédiées à la composition de WS ne peuvent ni exprimer ni traiter ce genre de problèmes puisqu'elles n'autorisent pas la création de nouveaux objets. Pour surmonter cette limite, nous

proposons une approche fondée sur les techniques de planification en IA où le langage de spécification du domaine est une extension du langage de spécification PDDL (Ghallab *et al.*, 1998) et le mécanisme de composition des Web Service s'appuie sur deux mécanismes de planification qui sont *Tree-search* et *Graphplan*.

4 Cadre formel

Notre cadre formel est basé sur une extension des techniques de planification (Ghallab *et al.*, 2005) pour pouoir créer et éliminer des objets dans l'environnement lors de l'exécution des actions (services). Contrairement aux approches classiques où l'état est défini comme un ensemble de prédicats, l'état de notre domaine est défini par un ensemble d'objets, de propriétés et de relations entre ces objets, permettant la création et l'élimination d'objets, l'attribution de nouveaux prédicats aux objets ainsi que la définition de nouvelles relations entre eux.

4.1 Définitions Préliminaires

Le domaine $D = (C, P)$ définit un ensemble de WS où $C = (WS_1, WS_2, ..., WS_n)$ une communauté de WS et P= $\{p_1, p_2, ..., p_n\}$, un ensemble de prédicats pour spécifier les propriétés des objets et les relations entre eux.

Dans l'exemple précédent, C= (WS_1, WS_2, WS_3, WS_4, WS_5) et P= {(en $?F$),($merge$ $?F1$ $?F2$ $?F$),($jpeg$ $?I$), ($mp3$ $?T$),... } où $?F$, $?I$ et $?T$ sont respectivement un fichier texte, une image, et un fichier audio. Le prédicat (en $?F$) signifie que $?F$ est en anglais, ($merge$ $?F1$ $?F2$ $?F$) est une relation qui signifie que $?F$ est le résultat de la fusion de $?F1$ et de $?F2$.

Définition 1
Un état $q = (O, P)$ du plan d'exécution est défini par un ensemble d'objets O avec leur types, et un ensemble de prédicats P exprimant les proprietés de ces objets ou les relations entre eux.

Dans l'exemple de la section 3, l'état initial est spécifié par :

$$q_0 = (\{(F1 : file), (F2 : file)\}, \{(doc\ \ F1), (en\ \ F1), (latex\ \ F2), (fr\ \ F2)\}$$

où $F1$, $F2$ sont des objets (fichiers textes) et $file$ est un type. doc, en, $latex$, et fr sont des propriétés.

Définition 2
un Web Service WS_i est défini par WS_i = (T_i, A_i, S_i) tel que :
- *T_i est le type du service.*
- *A_i est l'ensemble d'attribus du Web Service.*
- *$S_i = (s_{i1}, ..., s_{im})$ est l'ensemble de services composant WS_i.*

Dans l'exemple, WS_1 = (T_1, A_1, S_1)= ({translation},{IP='127.127.0.41'},{$fr2en$, $en2ar$, $en2fr$}).

Définition 3
Un service s_{ik} dans un WS_i est défini par $s_{ik} = (Pin_{ik}, Pout_{ik}, Pinout_{ik}, Prec_{ik}, Effects_{ik})$ où :
- *$Pin_{ik} = \{pin_{ik1}, pin_{ik2}, ...\}$ est l'ensemble d'objets en entrée du service (image par exemple).*
- *$Pout_{ik} = \{pout_{ik1}, pout_{ik2}, ...\}$ est l'ensemble d'objets en sortie du service.*
- *$Pinout_{ik} = \{pinout_{ik1}, pinout_{ik2}, ...\}$ est l'ensemble d'objets entrée-sortie du service.*
- *$Prec_{ik}$ est l'ensemble des conditions à satisfaire par tous les objets de Pin et de $Pinout$ du service.*
- *$Effects_{ik}$ est l'ensemble des effets d'exécution du service dans l'état courant du domaine.*

Il existe deux types d'effets dus à l'exécution d'un service : les effets implicites et les effets explicites. Les effets implicites consistent à éliminer tous les objets en entrée et à créer de nouveux objets (le nom d'un nouvel objet commence par #) pour chaque paramètre en sortie (l'exécution d'un service conduit à la réservation de ses objets $input$ et $inout$, mais avec la possibilité de changer leurs prédicats et leurs relations). Les effets explicites consistent à éliminer les prédicats de l'ensemble $Effect^-(s_{ik})$ et d'ajouter les prédicats de l'ensemble $Effect^+(s_{ik})$, où $Effect(s_{ik}) = Effect^+(s_{ik}) \cup Effect^-(s_{ik})$.

Considérons de nouveau l'exemple introduit dans la section 3 : le service $mergepdf$ du WS WS_3 est défini comme suit :

- Pin_3= { $(F1 : file)$,$(F2 : file)$ }.
- $Pout_3$= {$(F : file)$ }.
- $Pinout_3$= { }.
- $Prec_3$={$(pdfF1)$, $(pdfF2)$}.
- $Effect_3^-$={$(pdf\ F1)$,$(pdf\ F2)$}.
- $Effect_3^+$={$(pdf\ F)$}.

Définition 4
Un plan est défini par une séquence d'ensembles de services où chaque ensemble est un plan partiel. Formellement $\Pi =< \pi_1, \pi_2, ..., \pi_n >$ *est un plan où* $\forall i \in [1..n], \pi_i = (s_{i1}, ..., s_{in})$ *est un plan partiel de services indépendants où chaque* s_{ik} *est instancié avec des objets du domaine.*

L'exécution du plan $\Pi = \langle \pi_1, \pi_2, ..., \pi_n \rangle$ consiste en l'exécution ordonnée des plans partiels $\pi_i, i \in [1..n]$. Toutefois,les services d'un plan partiel π_i peuvent être exécutés dans un ordre quelconque, même en parallèle.

Un plan solution du problème introduit dans la section 3 est $\Pi =< \pi_1, \pi_2, \pi_3, \pi_4, \pi_5 >$ où :
- π_1=$(fr2en\ [F1], en2ar\ [F2])$.
- π_2=$(en2ar\ [F1], doc2pdf\ [F2])$.
- π_3=$(latex2doc\ [F1])$
- π_4=$(doc2pdf\ [F1])$.
- π_5=$([\#F_0 file] = mergepdf[F1, F2])$.

Définition 5
Un service s *est* **executable** *dans un état* q *si et seulement si :*
- *pour chaque object* $o \in \{P_{in} \bigcup P_{inout}\}$ *de* s*, il existe une variable dans l'état* q *qui est du même type que* o*.*
- *l'ensemble des préconditions sur les objets* P_{in} *et les objets* P_{inout} *sont toutes satisfaites.*

Dans l'exemple de la section 3, $doc2pdf[F1]$ est un service exécutable dans l'état initiale. $F1$ est de type $file$ et de format pdf. Après l'exécution d'un service, l'état du monde change. De nouveaux objets sont générés (les objets out de l'exécution du service) et de nouveaux prédicats sont attribués à ces derniers. D'autres mises à jour seront faites pour tous les objets de P_{inout} résultant des effets de ce service.

Définition 6
Une requête $R = (D, q_0, g)$ *est définie pour un domaine* D *de WS par l'état initial* q_0 *et l'état but* g*. L'état initial (resp. final) est defini par un ensemble d'objets* O *et un ensemble* P *de prédicats associés .*

Dans l'exemple précédant, l'état initial q_0 et l'état but g sont définis comme suit :

$$q_0 = [\{(F_1 : file), (F_2 : file)\}, \{(doc\ \ F_1), (en\ \ F_1), (latex\ \ F_2), (fr\ \ F_2)\}]$$

$$g = [\{(\#F_0 : file)\}, \{(pdf\ \ \#F_0), (ar\ \ \#F_0), (merge\ \ F_1\ \ F_2\ \ \#F_0)\}]$$

L'utilisation du symbole # devant le nom d'un objet signifie qu'il s'agit d'un objet généré (en sortie de l'ensemble des services exécutés). Tout objet ayant le même type et commençant par # peut le remplacer dans le domaine. On peut donc distinguer ces objets générés des objets définis dans l'état initial (ils ne commencent pas par le symbole #).

Définition 7
Soient $q_1 = (O_1, P_1)$ *et* $q_2 = (O_2, P_2)$ *deux états,*

$$q_1 = q_2 \text{ ssi } \begin{cases} \mid O_1 \mid \simeq \mid O_2 \mid & \forall o_i \in O_1 \\ & \text{si } o_i \text{ est un objet initial alors } o_i \in 0_2 \\ & \text{si } o_i \text{ est un objet généré alors } \exists o_j \in O_2 \text{tel que } type(o_j) = type(o_i) \\ \\ \mid P_1 \mid \simeq \mid P2 \mid & \forall (p_i\ \ o_1, \ldots, o_n) \in O_1 \exists (p_j\ \ o_1^{'}, \ldots, o_n^{'}) \in O_2 \text{tel que} \\ & \text{si } o_i \text{ est un objet intial alors } o_i = o_i^{'} \\ & \text{si } o_i \text{ est un objet généré alors } o_i^{'} \text{ est un objet généré} \end{cases}$$

Considérons les trois états suivants :

- $q_1 = [\{(\#F0 : file), (F1 : file)\}, \{(en\ \ \#F0), (ar\ \ F1)\}]$,
- $q_2 = [\{(\#F : file), (F1 : file)\}, \{(en\ \ \#F), (ar\ \ F1)\}]$,
- $q_3 = [\{(\#F0 : file), (F : file)\}, \{(en\ \ \#F0), (ar\ \ F)\}]$

Nous avons $q_1 = q_2$, mais $q_1 \neq q_3$ et $q_2 \neq q_3$

5 Algorithme de planification

Nous avons implémenté deux algorithmes pour construire la solution à notre problème de composition de WS. Le premier est basé sur l'algorithme *Tree-search* et le second sur la méthode *Graphplan*.

5.1 Algorithme de composition de WS fondé sur la méthode *Tree-search*

L'idée de base de l'algorithme de recherche *Tree-search* est d'appliquer à partir de l'état initial tous les services exécutables. Par conséquent, développer un état conduit à l'obtention d'un ensemble de nouveaux états. Sur la base de la stratégie de parcours de *Tree-search*, on choisit l'un des états non développés pour le développer et obtenir un plan séquentiel de services. Ensuite, nous transformons ce plan en une série séquentielle de plans partiels (ensemble des services indépendants) $\Pi = \langle \pi_1, \pi_2, ..., \pi_n \rangle$.

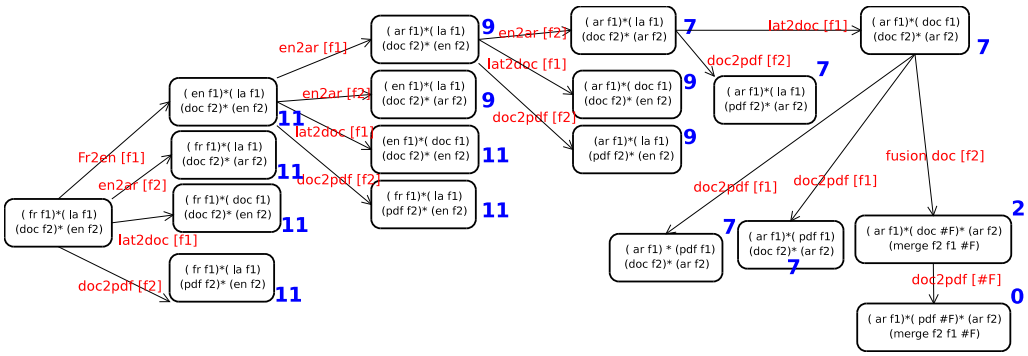


FIG. 1 – Tree-search avec Heurestique

L'algorithme de transformation (Algorithme 1) prend comme entrée un plan séquentiel de services, et le transforme en un plan séquentiel d'ensembles de services tout en se basant sur la dépendance des services (définition 9). Initialement, Π contient le plan partiel π_1 qui contient le premier service. Pour le reste des services s_k, on teste pour chacun d'entre eux s'il existe un plan partiel $\pi_i \in \Pi = \langle \pi_1, \pi_2, ..., \pi_f \rangle$ tel que tous ses services sont indépendants de s_k, alors on ajoute s_k dans π_i. Dans le cas contraire, on crée un nouveau plan π_{f+1} et on lui ajoute s_k.

Exemple : dans la figure 1 le plan séquentiel obtenu est $\{(fr2en\ [F1]), (en2ar\ [F1]), (en2ar\ [F2]), (latex2doc\ [F1]), \#F_O = (mergedoc\ [F1]\ [F2]), (doc2pdf\ [\#F_0])\}$. En appliquant l'algorithme de transformation, nous obtenons :

- $\pi_1 = (fr2en\ [F1], en2ar\ [F2])$.
- π_2=$(en2ar\ [F1])$.
- π_3=$(latex2doc\ [F1])$.
- π_4=$([\#F_0 file] = mergedoc[F1, F2])$.
- π_5=$(doc2pdf\ [\#F_0])$.

5.1.1 Optimisation de l'algorithme de *Tree-search* : introduction d'une heuristique

Au cours du parcours de l'arbre de recherche dans *Tree-search*, deux facteurs jouent un rôle important dans la diminution de l'espace de recherche et du temps d'exécution de l'algorithme qui sont le choix du service à appliquer et celui du nœud à développer. Nous définissons une heuristique pour le choix du noeud à développer lors de l'exécution de l'algorithme de *Tree-search*. L'idée de cette heuristique est de définir une distance entre un noeud courant et le noeud but. A chaque itération, on choisit le noeud le plus proche du but pour le développer. Cette distance dépend de l'ensemble des objets des deux nœuds et de leur prédicats.

Algorithme 1 : Algorithme de transformation du plan séquentiel des services en un plan séquentiel d'ensembles de services.

```
TransformerPlan(Π_in) :(Π_out)
n ⟵ taille(Π_in)
serviceCourant ⟵ service(1,Π_in)
nouveauNiveau(serviceCourant,Π_out)
i ⟵ 2
while i < n do
    serviceCourant ⟵ service(i,Π_in)
    π_final ⟵ niveau(Π_out,taille(Π_out))
    if depend(serviceCourant,π_final)
    then
        nouveauNiveau(serviceCourant,Π_out)
    else
        j ⟵ taille(Π_out)-1
        while j>0 do
            π_courant ⟵ niveau(Π_out,j)
            if depend(serviceCourant,π_courant)
            then ajouterservice(serviceCourant,Π_out,j+1)
            j ⟵ 0
        end
    end
    i ⟵ i+1
    if j=0 then ajouterService(serviceCourant,Π_out,1)
end
return Π_out
```

Définition 8
Une heuristique est définie pour deux états $N_1 = (O_1, P_1)$ *et* $N_2 = (O_2, P_2)$ *(noeuds de l'arbre) du domaine par* ***heuristique***(O_1, O_2)= $(|O_1| + |O_2| - 2 * |O_1 \cap O_2|)$ + $(|P_1| + |P_2| - 2 * |P_1 \cap P_2|)$

Cette heuristique est facilement appliquée lorsque les nœuds ne contiennent pas d'objets générés. Dans le cas contraire, pour chaque objet généré d'un nœud courant n, on lui attribue le nom d'un objet généré de même type appartenant au nœud but. Si après ce renommage, le nœud final n'est pas atteint, on calcule la distance décrite ci-dessus. Si après tous les renommages possibles, le nœud but n'est pas atteint, on attribue à n la distance minimale des renommages possibles. Cette heuristique n'augmente pas la complexité de l'algorithme de recherche puisque dans l'algorithme de base il est nécéssaire de renommer pour chaque nœud courant, les objets générés en fonction des objets générés du nœud but pour vérifier si celui-ci est atteint.

5.2 Exemple

Soient les 4 états suivants :
- $init = (\{(F1 : file), (F2 : file)\}, \{(fr\ \ F1), (latex\ \ F1), (doc\ \ F2), (an\ \ F2)\}$
- $S1 = (\{(F1 : file), (F2 : file)\}, \{(latex\ \ F1), (en\ \ F1), (pdf\ \ F2), (an\ \ F2)\}$
- $S2 = (\{(\#F0 : file)\}, \{(ar\ \ F1), (ar\ \ F2), (doc\ \ \#F0), (merge\ \ F2\ F1\ \#F0)\}$
- $goal = (\{\#F0file)\}, \{(ar\ \ F1) * (ar\ \ F2) * (pdf\ \ \#F0) * (merge\ \ F2\ F1\ \#F0)\}$

Nous avons :
- heuristique(init,goal)=(2+1-2*0)+(4+4-2*0)= 11
- heuristique(init,S1)=(2+2-2*2)+(4+4-2*2) = 4
- heuristique(S2,goal)=(1+1-2*1)+(4+4-2*3) = 2

5.3 Algorithme de composition de WS fondé sur la méthode *Graphplan*

La technique *Graphplan* introduit un espace de recherche très puissant s'appuyant sur le graphe de planification. Elle permet de construire l'espace des états founissant un plan comme une séquence d'actions (dans notre cas de services), et l'espace de plans exprimant un plan comme un ensemble de services partiellement ordonnés (Ghallab *et al.*, 2005). Une contribution importante du planificateur *Graphplan* représente

la relaxation de l'analyse de l'accessibilité. Cette approche fournit une condition incomplète de l'accessibilité par le biais du planificateur. Cette condition n'est pas suffisante pour trouver une solution, mais elle a une faible complexité puisque la construction du graphe de planification est polynomiale en terme de la taille de graphe.

L'idée de base de cette technique est de considérer à chaque niveau du graphe, non pas un état individuel, mais pour une première approximation, l'union des prédicats de plusieurs états. Un graphe de planification est un graphe construit en niveaux (FiG.2) : les arcs ne sont autorisés que d'un niveau à un autre. Un noeud dans le niveau 0 du graphe correspond à l'ensemble de prédicats P_0 caractérisant l'état initial q_0 du problème. Le niveau i contient deux sous-niveaux : le sous-niveau d'actions A_i et le sous-niveau de prédicats P_i, et ainsi de suite.

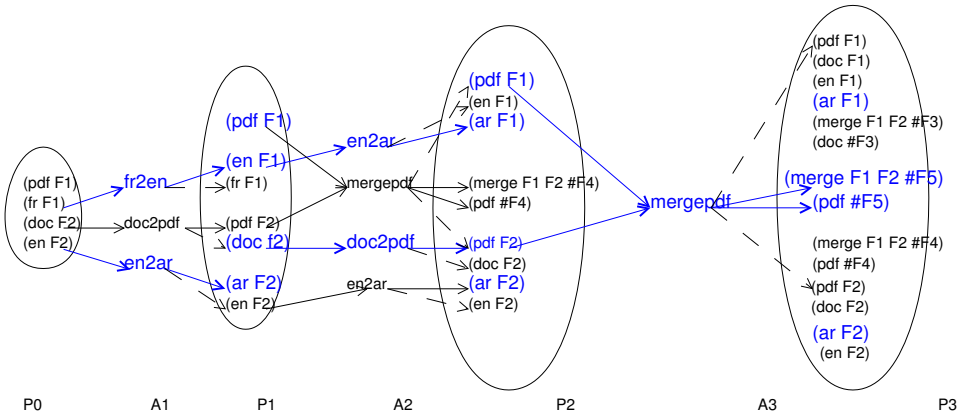


FIG. 2 – Exécution de Graphplan

Avant de poursuivre la présentation de notre approche de composition fondée sur cette technique, nous devons définir les exclusions mutuelles *mutex* pour les services et les prédicats. Pour cela, nous introduisons la notion d'indépendance de services comme suit :

Définition 9
Deux services s_1 et s_2 sont indépendants si et seulement si :
- $\{Pinout_1 \cup Pout_1\} \cap \{Pin_2 \cup Pinout_2\} = \emptyset$.
- $\{Pinout_2 \cup Pout_2\} \cap \{Pin_1 \cup Pinout_1\} = \emptyset$.

Exemple : $fr2en[F1]$, $en2ar[F2]$ sont indépendants, tandis que $([\#F_0 file] = mergepdf[F1, F2])$ et $en2ar[F2]$ sont dépendants.

Définition 10
Deux services $s_1 \in A_i$ et $s_2 \in A_i$ sont mutex ssi :
- *s_1 et s_2 sont dépendants, ou*
- *si une précondition de s_1 est mutex avec une précondition de s_2.*

Deux préconditions $p \in P_{i-1}$ et $p' \in P_{i-1}$ sont mutex si chaque service dans A_i admettant p comme un effet positif est mutex avec tout service produisant p', et il n'y a aucun service dans A_i qui produit p et p'.

L'algorithme Graphplan est itératif et permet de construire le graphe en profondeur en y rajoutant un nouveau niveau à l'espace de recherche à chaque itération. Le développement du graphe se poursuit jusqu'à obtenir un niveau i tel que P_i contient tous les prédicats du but ou alors $P_i = P_{i-1}$. Lorsque le niveau atteint contient tous les prédicats du but, on effectue à partir de ce niveau un parcours en arrière pour extraire un plan valide c'est-à-dire ne contenant pas d'actions (services) mutex. Si c'est le cas ce plan sera le plan solution, sinon on continue le développement de l'arbre sauf si $P_i = P_{i-1}$. Dans le cas ou le niveau i atteint est tel que $P_i = P_{i-1}$ et aucun plan ne peut être extrait à partir de P_{i-1} , on parle alors de situation d'échec.

La figure 2 représente l'exécution de l'algorithme de Graphplan pour l'exemple 3 avec comme état initial ($q_0 = (\{(F1 : file), (F2 : file)\}, \{(pdf \quad F1), (fr \quad F1), (doc \quad F2), (en \quad F2)\}$ (les lignes pointillées correspondent aux effets négatifs, et tous les arcs ne sont pas affichés). En effet, le but $g = [\{(\#F_0 : file)\}, \{(pdf \ \#F_0), (merge \ F_1 \ F_2 \ \#F_0), (ar \ \#F_1), (ar \ \#F_2)\}]$ est dans $P3$ sans mutex. La seule action dans $A3$ accomplissant g est $\pi_3 = [mergepdf]$. Dans le niveau 2, le nouveau but $g_1 = \{goal -$

$effects(\pi_3) + preconditions(\pi_3)\}$ devient $\{(pdf\ \ F1)(ar\ \ F1)(pdf\ \ F2)(ar\ \ F2)\}$. $(ar\ \ F1)$ et est accompli par $en2ar$ et $(pdf\ \ F2)$ est accompli par $doc2pdf$ d'où $\pi_2 = [en2ar, doc2pdf]$.

Dans le niveau 1, le nouveau but $g2$= $\{goal - effects(\ \pi_2)$ + $preconditions(\pi_2)\}$ devient $\{(pdf\ \ F1)(en\ \ F1)(pdf\ \ F2)(ar\ \ F2)\}$, $(en\ \ F1)$ et est accompli par $fr2en$, et $(ar\ \ F2)$ est accompli par $en2ar$.

Le niveau 0 est atteint avec succès (niveau 0 contient les predicats de l'état initial).

5.4 Heuristiques avancés

Les techniques de grille de calcul ont été utilisées dans (J. Blythe, 2003) pour générer automatiquement des workflows de tâche. Cette approche est dédiée à traiter les problèmes qui nécessitent l'utilisation des ressources distribuées sur le réseau pour résoudre des tâches à grande échelle telles que le calcul scientifique. Deux planificateurs ont été utilisés dans cette approche : PRODIGY (Veloso *et al.*, 1995) et FF (Hoffmann & Nebel, 2001).

Dans PRODIGY, l'espace de recherche est représenté comme dans *Tree-search* et un plan non complet se compose de deux parties, plan-tête et plan-queue. Le plan-queue est construit par un algorithme d'ordre partiel de chaînage arrière, qui part du but G et ajoute les opérateurs, un par un, pour accomplir les préconditions d'autres opérateurs qui sont fausses dans l'état courant.

Des stratégies basées sur des techniques d'apprentissages sont utilisées dans PRODIGY pour l'exploration de l'espace de recherche. Ces stratégies utilisent des règles de contrôle pour orienter le choix de l'état à développer et de l'opérateur à appliquer. Ces techniques visent à reduire le temps de planification, améliorer la qualité du plan solution, et raffiner le domaine de connaissances.

PRODIGY voit les épisodes de planification comme une opportunité d'apprendre des règles de contrôle pour améliorer à la fois les performances du planificateur et la qualité des plans produits. PRODIGY suppose que l'appreneur est formé avec des problèmes simples pour lesquels le planificateur peut explorer l'espace de tous les plans. Les données d'entrée de l'appreneur sont : le domaine, l'ensemble des problèmes d'apprentissages, la mesure de la qualité du plan.

Dans le planificateur FF, l'espace de recherche est représenté comme dans Graphplan. FF utilise une heuristique qui estime la distance au but en ignorant les effets de supprimer des prédicats. FF introduit une nouvelle stratégie qui combine la recherche systématique avec une technique de hill-climbing. De plus, il montre comment d'autres heuristiques puissantes peuvent être extraites et utilisées pour élaguer l'espace de recherche.

Nous comptons étendre notre travail en étudiant ces approches en détail pour essayer de les appliquer dans le domaine de composition des Web Services traitant des requêtes implicites ce qui n'est pas le cas en ce moment dans ces approches.

6 Implémentation

En implémentant les algorithmes *Tree-search* et *Graphplan*, nous démontrons que notre nouvelle approche de la composition des WS sous requêtes implicites est réalisable. La première étape est de définir un langage de spécification du domaine de planification à savoir l'ensemble des WS disponibles, l'ensemble des prédicats ainsi que l'ensemble des types des objets. Dans notre implémentation, nous utilisons une partie du langage PDDL (Ghallab *et al.*, 1998) que nous adaptons à notre modèle. En effet, nous utilisons une partie de la forme générale de la spécification des actions et des domaines dans PDDL pour définir nos services. Notre extension consiste à ajouter le symbole # pour désigner les objets générés et permettre aux services d'exprimer l'ajout et la suppression d'objets lors d'exécution.

PDDL est destiné à décrire l'état physique d'un domaine, à savoir ses prédicats, actions (ou services), la structure des actions composées et enfin les effets des actions. Les objectifs de développement de ce language standard est d'encourager l'évaluation empirique des performances du planificateur et le développement d'un ensemble de problèmes dans une notation comparable. En général, la plupart des planificateurs ne manipulent pas la totalité du langage PDDL. Toutefois, il est souvent nécessaire d'étendre les notations PDDL en fonction des domaines d'application.

6.1 Exemple de specification de service et de requête en PDDL

Revenons à l'exemple : la définition du service mergepdf est comme suit :

		stratégie					
		profondeur		largeur		Heuristique	
Problème	nbr d'objets	nb nœuds	taille du plan	nb nœuds	taille du plan	nb nœuds	taille du plan
P1	1	5	4	9	4	5	4
P2	2	7	6	270	6	7	6
P3	2	11	4	15	4	5	4
P4	3	51	10	$\bowtie$	$\bowtie$	12	11
P5	4	1279	$\oslash$	1279	$\oslash$	1279	$\oslash$

$\oslash$ pas de solution , $\bowtie$ solution non trouvé

TAB. 1 – Resultat de l'exécution de Tree-search

```
(:action mergepdf
:in_parameters (?f5 -file ?f6 -file)
:in_out_parameters ()
:out_parameters (?f7 -file)
:precondition (pdf ?f5)*(pdf ?f6)
:effect (pdf ?f7)*(merge ?f5 ?f6 ?f7))
```

La specification de l'état initial et de l'état but est comme suit :

```
init= (parameters=(F1 file)*(F2 file)
       predicates = (fr F1)*(latex F1)*(doc F2) * (an F2))
goal= (parameters=(\#F0 file)
       predicates=(ar F1)*(ar F2)*(pdf \#F0)*(merge F2 F1 \#F0))
```

6.2 Implémentation de *Treesearch*

Nous avons testé l'algorithme *Treesearch* pour cinq exemples qui contiennent de nombreux objets et types de variables. Dans le tableau 1 la première colonne représente les différents problèmes traités : le problème P2 est celui introduit dans la section 6.1 et la seconde colonne indique le nombre d'objets initiaux des différents problèmes. Les colonnes suivantes décrivent le nombre de noeuds développés et la taille du plan en utilisant la stratégie en profondeur, la stratégie en largeur et la stratégie utilisant l'heuristique. Nous avons 16 services disponibles dans le domaine (illustrés dans la section3). À partir de ces résultats, nous pouvons constater que la stratégie utilisant l'heuristique est la plus efficace car nous obtenons un plan en développant un petit nombre de noeuds. Toutefois, la stratégie en largeur permet d'obtenir des solutions pour les mêmes problèmes, mais en développant un grand nombre de noeuds. Pour le problème P5, plus de 1279 noeuds ont été développés pour constater qu'il n'y a aucun plan solution ($\oslash$) même en utilisant la stratégie de l'heuristique. Nous pensons, pour remédier à ce problème, à une autre heuristique permettant de sélectionner le meilleur service à appliquer.

6.3 Implémentation de *Graph Plan*

L'algorithme Graphplan est plus adapté aux problèmes simples (ne contenant pas un grand nombre d'objets et de services). En fait, en appliquant les techniques étendues de *Graphplan*, on obtient une explosion combinatoire due à l'exécution de services qui créent de nouveaux objets à chaque niveau. Par exemple, dans le cas du problème P5 du tableau 1 qui contient 4 fichiers, après le développement du graphe nous obtenons dans le cinquième niveau plus de 50 fichiers objets et plus de 120 prédicats. Ceci est dû au fait qu'on n'élimine ni les prédicats ni les objets des niveaux lors du développement du graphe.

En outre, afin de tester si nous parvenons à un niveau qui contient l'état but, nous devons calculer tous les prédicats et les services mutex pour chaque niveau. Ce qui nécessite beaucoup d'espace mémoire et beaucoup de temps CPU (pour 120 prédicats dans un niveau on teste 120*119 paires de pédicats).

Nous concluons que l'utilisation de Graphplan n'est pas efficace dans ce genre de problème, parce que nous avons une double complexité (mémoire et temps d'exécution) très élevée.

7 Conclusion et perspectives

Dans cet article, nous donnons une vue étendue du problème de composition de Web Service en le modélisant comme un problème de planification. Nous proposons un modèle de service pour répondre à des

problèmes de composition exigeant la création et l'élimination d'objets comme des effets de l'exécution d'un service. Nous avons également surmonté les limites des autres approches, en donnant une définition dynamique et distribuée de notre domaine. Cela nous permet d'ajouter, de supprimer et/ou de remplacer les services sans recalculer une autre partie du domaine. Enfin, notre modèle surmonte la limite des plans pré-définis par la définition implicite de la requête uniquement par le biais d'un état initial et un état but. Nous avons présenté des résultats encourageants sur l'utilisation de *Tree-search*, même si des extensions sont nécessaires pour trouver de nouvelles stratégies de composition plus efficaces.

Nous comptons par la suite introduire une problématique de diagnostic d'exécution du plan de WS. Nous aurons à piloter l'exécution du plan et en déduire l'état courant (éventuellement défectueux) d'exécution du plan, qui sera utilisé pour réparer la faute en re-planifiant la composition en affectant l'état défectueux comme état initial.

A partir de là, nous voulons définir de nouveaux paramètres pour la notion de fiabilité des plans. Le degré de fiabilité d'un plan dépend des état défectueux réparables qui peuvent se produire au cours de l'exécution du plan. En calculant cette métrique on peut distinguer les plans les plus fiables parmi plusieurs plans susceptibles de répondre à la même requête.

Enfin, nous voulons renforcer notre modèle, à travers la notion de réservation des ressources : une fois le plan trouvé, il faut, avant de l'exécuter, réserver les services qui sont nécessaires sur Internet, afin d'éviter des conflits avec d'autres plans exécutés en parallèle sur la communauté des WS.

Références

ANDREWS T., CURBERA F. & DOLAKIA H. (2003). Business process execution language for web services. In *Technology Report, www.download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf*.

BERTOLI P., CIMATTI A., PISTORE M., ROVERI M. & TRAVERSO P. (2001). MBP : a model based planner. In *the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information, Seattle.*

CHINNICI R., GUDGIN M. & MOREAU J.-J. (2004). Web services description language. In *Technology Report, http ://www.w3.org/TR/2004/WD-wsdl20-20040326/.*

GHALLAB M., HOWE A., KNOBLOCK C., MCDERMOTT D., RAM A., VELOSO M., WELD D. & WILLKINS D. (1998). PDDL the planning domain definition language. In *AIPS'98, The Fourth International Conference on Artificial Intelligence Planning Systems).*

GHALLAB M., NAU D. & TRAVERSO P. (2005). Automated planning theory and practice. In *Morgan kaufman Publishers*.

GOTTSCALK K. (2000). Web services architecture overview : the next stage of evolution for e-business. In *IBM Web Services architecture team, ftp ://www6.software.ibm.com/software/developer/library/wovr.pdf*.

HOFFMANN J. & NEBEL B. (2001). The FF planning system : Fast plan generation through heuristic search. In *Journal of Artificial Intelligence Research*.

J. BLYTHE, E. DEELMAN Y. G. C. K. A. A. G. M. . K. V. (2003). The role of planning in grid computing. In *The International Conference on Automated Planning and Scheduling (ICAPS).*

LAZOVIK A., AIELLO M. & PAPAZOGLOU M. (2003). Planning and monitoring the execution of web service requests. In *the first International Conference on Service-Oriented Computing (ICSOC'03)*.

MCILRAITH S. & SON T. (2002). Adapting GOLOG for composition of semantic web services. In *KR02, International Conference on Principles of Knowledge Representation and Reasoning*.

NAU D., AU T., ILGHAMI O., KUTER U., MURDOCK J. W., WU D. & YAMAN F. (2003). SHOP2 : An HTN planning system. In *J. Artif. Intell. Res. (JAIR)*, p. 379–404.

PISTORE M., BERTOLI P., BARBON F., SHAPARAU D. & TRAVERSO P. (2004). Planning and monitoring web service composition. In *AIMSA04, Artificial Intelligence : Methodology, Systems, and Applications*.

PISTORE M. & TRAVERSO P. (2001). Planning as model checking for extended goals in non-deterministic domains. In *IJCAI - International Joint Conferences on Artificial Intelligence*, p. 479–486.

RUSSEL S. & NORVIG P. (2003). Artificial intelligence : A modern approach. In *Prentice Hall, Chapter 3 and 4*.

VELOSO M., CARBONELL J., PÉREZ A., BORRAJO D., FINK E. & BLYTHE J. (1995). Integrating planning and learning : The PRODIGY architecture. In *Journal of Experimental and Theoretical Artificial Intelligence*, p. 81–120.

WU D., SIRIN E., HENDLER J., NAU D. & PARSIA B. (2003). Automating DAML-S web services composition using SHOP2. In *International Conference on Automated Planning and Scheduling , ICAPS 2003.*

Composition dynamique de systèmes interactifs : l'alliance de l'IHM et de la Planification

Yoann Gabillon[1], Gaëlle Calvary[1], Humbert Fiorino[1]

[1] Laboratoire Informatique de Grenoble (LIG), Université Joseph Fourier,
{yoann.gabillon, gaelle.calvary, humbert.fiorino}@imag.fr

Résumé : Aujourd'hui, une Interface Homme-Machine (IHM) est le fruit de transformations souvent mentales, appliquées à une représentation abstraite de l'IHM appelée modèle de tâches. Ces transformations font des compromis entre le contexte d'usage courant (<Utilisateur, Plate-forme, Environnement>) et les propriétés d'utilisabilité à satisfaire (propriétés ergonomiques). En informatique ambiante, le contexte d'usage devient variable et imprévisible. Il est donc nécessaire de doter le système de capacité de raisonnement (modèles et transformations) pour lui permettre de conformer l'IHM à ce contexte dynamique. Notre étude explore la planification automatique pour la construction d'un système interactif adapté à un objectif utilisateur (une représentation abstraite partielle de l'IHM) et un contexte d'usage donnés. L'Ingénierie Dirigée par les Modèles (IDM) est utilisée comme passerelle entre ces deux communautés. L'article présente le premier prototype, le cas d'étude et la mise en correspondance entre IHM et planification.

Mots-clés : Composition dynamique, contexte d'usage, IHM, planification, IDM.

1. Introduction

La décomposition fonctionnelle minimale d'un système interactif distingue le Noyau Fonctionnel (NF) de l'Interface Homme-Machine (IHM) (Fig. 1). Le NF regroupe l'ensemble des traitements indépendamment de toute représentation à l'utilisateur. L'IHM fait les choix de représentation compte tenu du contexte d'usage ciblé et des propriétés ergonomiques à satisfaire. Par contexte d'usage, on entend l'utilisateur (ses caractéristiques, compétences, préférences, etc.), la plate-forme d'exécution (ses capacités de calcul, de communication ainsi que les ressources d'interaction disponibles en entrée et en sortie), l'environnement physique/social (niveaux lumineux, sonore, etc.)

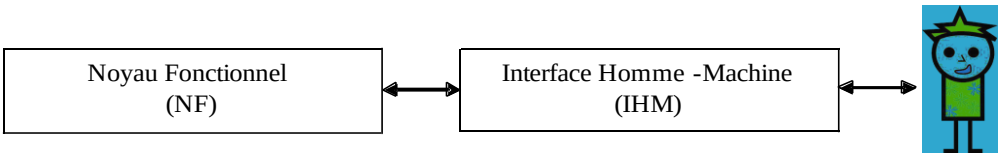


Fig. 1 - Décomposition fonctionnelle minimale d'un système interactif.

C'est l'analyse des besoins qui permet de cerner la couverture fonctionnelle à donner au NF. S'ensuit alors un processus de développement en V, spirale ou autre (Balbo, 1994). L'architecture logicielle en est une étape clé : elle couvre l'identification des composants, processus et ressources physiques (dont les processeurs) ainsi que l'allocation des fonctions (NF et IHM) aux composants, des composants aux processus et des processus aux ressources physiques (Coutaz, 2001). Si jusqu'ici ces identifications (fonctions, composants, processus et ressources physiques) et allocations (fonctions aux composants, …, processus aux ressources physiques) étaient décidées à la conception, elles doivent désormais être dynamiques pour répondre à la variabilité et imprévisibilité du contexte d'usage en informatique ambiante.

Dès lors que l'utilisateur est mobile, son contexte d'usage (Florins, 2006) (plate-forme, environnement) peut varier dynamiquement au fil par exemple de l'arrivée ou du départ de ressources. Ses besoins peuvent également émerger opportunément, suite par exemple à une situation imprévue. Il

devient, en conséquence, nécessaire de savoir composer dynamiquement des systèmes interactifs qui : (1) remplissent les objectifs fonctionnels de l'utilisateur (« faire telle tâche »), (2) aient les bonnes propriétés du point de vue de l'interaction homme–machine (temps de calcul, présentation), (3) soient compatibles avec le contexte d'usage courant. Cet article explore la planification automatique pour la composition dynamique de systèmes. L'étude est originale :

- beaucoup de travaux étudient la composition statique de systèmes interactifs (composition de services Web (MacKenzie *et al.*, 2006), composition d'IHM (Myers, 1993) mais aucun ne cible la composition dynamique ;
- A notre connaissance, les techniques de planification n'ont encore jamais été confrontées aux exigences de l'interaction homme-machine : partage de tâches entre l'homme et la machine avec des boucles de contrôle telles que l'itération, distribution de l'IHM sur les plates-formes disponibles, ergonomie du plan produit ;
- la description des IHM du point de l'ergonomie est inédite.

L'article présente nos travaux préliminaires : le cas d'étude, un premier prototype puis l'approche adoptée. L'article développe ensuite l'IDM comme passerelle entre IHM et planification. Cette mise en correspondance de disciplines permet d'identifier des convergences et divergences de différentes natures.

2. Cas d'étude et prototype MAPPING

Le cas d'étude est celui d'un chercheur (Bob) inscrit à une conférence. Bob arrive en retard à une session d'intérêt. Déçu, il reste dans le couloir, faute de place dans l'amphithéâtre. Il suit la présentation à distance grâce à un mur écran installé dans le couloir, mais au moment des questions, il aimerait se manifester. Il décide alors d'utiliser le système MAPPING (Models, Artificial Perception and Planning for Interactive systems Generation) pour pouvoir poser sa question au conférencier. Il spécifie son objectif - " I want to ask questions " - en langage naturel dans une fenêtre dédiée (Fig. 2).

L'objectif utilisateur (ex : poser une question) et la procédure qui lui est associée (une décomposition en sous-tâches par le biais d'opérateurs) sont représentés par une tâche d'un modèle de tâches. Un modèle de tâches est une représentation abstraite et canonique d'une IHM (ex : Identifier un utilisateur). Une tâche possède un nom, une catégorie (par exemple, tâche abstraite si elle est décomposée en sous-tâches de catégories hétérogènes, tâche mentale ou tâche d'interaction), un type (comparaison, édition, etc.) Les tâches sont liées entre elles par des opérateurs, donnant globalement lieu à un arbre des tâches. Les « opérateurs » peuvent être logiques (et, ou) ou temporels (séquence, entrelacement, etc.) Une tâche manipule des concepts. Ce sont les entités du domaine. Les tâches peuvent être décorées. Les décorations expriment classiquement la criticité d'une tâche (tâche irréversible), sa fréquence, son caractère itératif (la tâche peut être exécutée plusieurs fois par l'utilisateur) ou optionnel (tâche facultative). Des contraintes temporelles peuvent aussi être exprimées (durée minimale, maximale et moyenne).

Le système MAPPING analyse l'objectif utilisateur grâce au moteur de bon sens ConceptNet (Liu *et al.*, 2004). Il établit alors un modèle des tâches incomplet : dans l'exemple, il est réduit au seul objectif utilisateur. Aucune procédure (i.e., façon d'accomplir la tâche pour atteindre l'objectif) ne lui est associée. Grâce à un annuaire de services stockant des descriptions de systèmes interactifs en termes de tâches utilisateur offertes et de ressources d'interaction requises, MAPPING identifie deux procédures possibles pour atteindre le but : recourir à un outil de communication synchrone (par exemple, Messenger) ou asynchrone (par exemple, le courrier électronique) (Fig. 2). Les deux plans sont proposés à l'utilisateur. La Figure 3 présente le modèle de tâches simplifiéde « Poser Question ».

La figure 4 montre l'application de MAPPING à un exemple où la combinatoire est plus élevée : la tâche utilisateur est plus complexe (assister à des présentations sur un sujet donné) et la plate-forme plus

variée : réduite à un seul PC en figure 2, elle est ici étendue à un PC et un PDA, allumés ou éteints, branchés ou non à un clavier physique. L'utilisateur (anglophone) et l'environnement (sonore) restent fixés. Vingt-quatre plans sont trouvés.

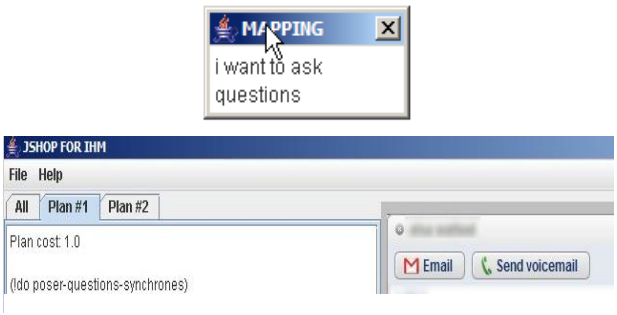


Fig. 2 – Spécification de l'objectif utilisateur en langage naturel. Obtention de deux plans selon le mode synchrone ou asynchrone de la question.

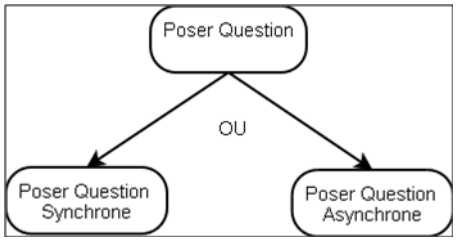


Fig. 3 – Modèle (Arbre) de tâche simplifié de « Poser Question ».

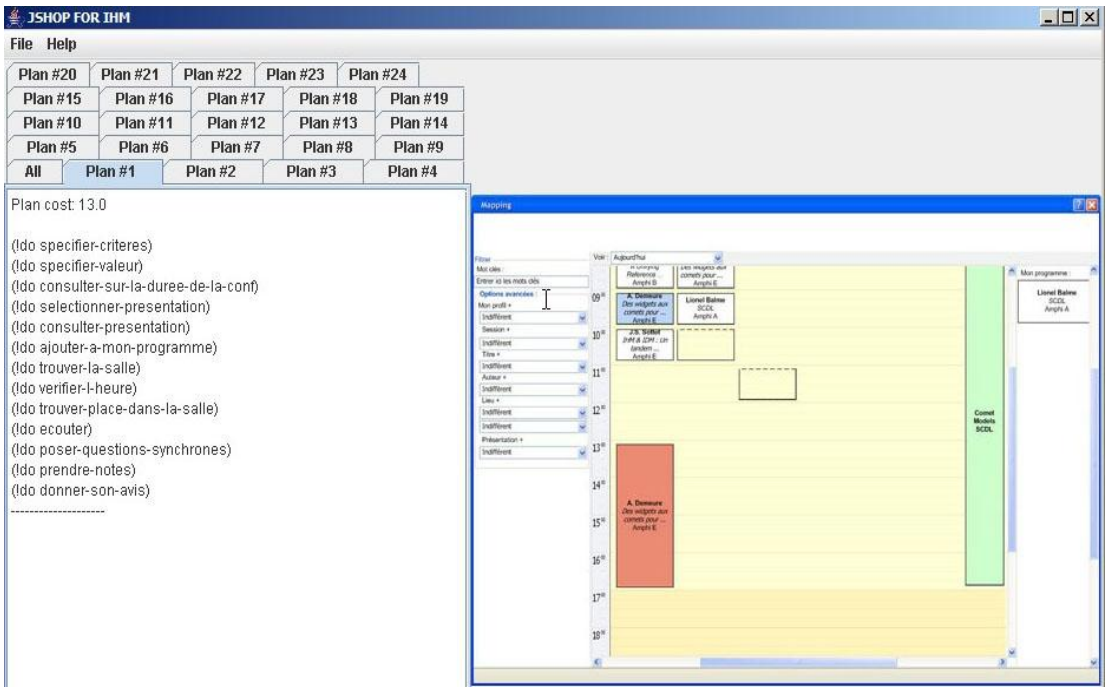


Fig. 4 - Vingt-quatre plans possibles pour l'élaboration du programme scientifique de Bob. L'IHM du plan sélectionné (#1) s'affiche en partie droite.

Le paragraphe suivant détaille l'approche adoptée.

3. Approche

Notre approche consiste à utiliser des algorithmes de planification pour composer des IHM. L'IDM (Ingénierie Dirigée par les Modèles) (Belaunde *et al.*, 2003) est le domaine de l'informatique mettant à disposition des outils, concepts et langages pour créer et transformer des modèles. L'IDM est utilisée comme passerelle entre les domaines IHM et planification. De façon plus précise, l'approche s'articule en cinq temps (Fig. 5) :

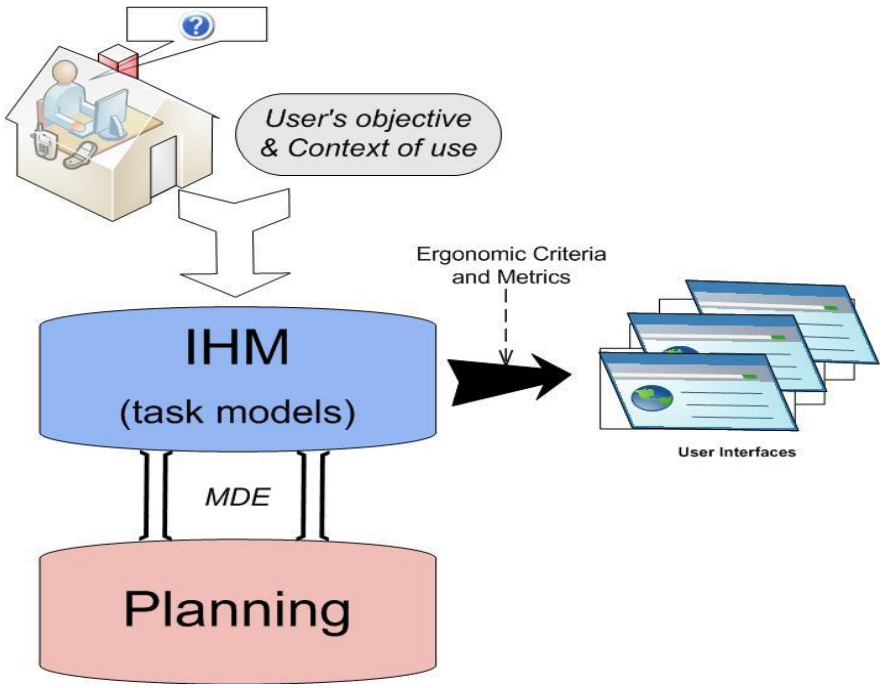


Fig. 5 - Approche suivie

1. Spécification du problème dans le domaine IHM : l'objectif utilisateur est analysé et reformulé en un modèle de tâches incomplet (l'analyse se fait via un outil appelé ConceptNet). Dans une approche classique, le modèle de tâches est entièrement spécifié à la conception.

2. Réécriture du problème en termes de planification (Nau *et al.*, 1998). : le modèle de tâches IHM est transformé en domaine HTN ; les ressources définissent l'état du monde.

3. Calcul de plans par un planificateur : le calcul s'appuie sur les composants réutilisables (modèles ou code relevant du NF ou de l'IHM). Dans MAPPING, une IHM est associée à chaque modèle de tâches. Les modèles et codes sont entreposés dans un annuaire de services. Le planificateur utilisé est JSHOP (Ilghami & Nau, 2003). JSHOP combine les modèles de tâches pour réaliser l'objectif utilisateur compte tenu des ressources disponibles. JSHOP renvoie la séquence d'actions correspondant aux tâches à effectuer.

4. Réécriture des plans en IHM : le plan est réécrit en modèle de tâches.

5. Transformation du modèle de tâches pour construire une IHM ergonomique : le modèle de tâches peut être restructuré par ajout de tâches abstraites (ie. de tâches servant à structurer l'arbre de tâches et donc l'IHM) pour typiquement éviter des « râteaux » (arbre de tâches de profondeur 1) qui aboutiraient à des IHM de petites tailles, comportant peu d'informations, et insatisfaisantes d'un point de vue ergonomique ; des concepts peuvent aussi être factorisés pour une meilleure prise en compte des critères d'ergonomie (Scapin & Bastien, 1997). Par exemple, il ne serait ergonomiquement adéquat de faire apparaître l'heure à deux endroits différents d'une IHM.

MAPPING est un démonstrateur « en largeur », il abordel'ensemble des problématiques de notre approche mais ne les résout pas toutes. De nombreuses questions restent ouvertes tant en description (quels langages de tâches, de ressources etc. ?) qu'en adaptation des algorithmes de planification. Aujourd'hui, notre travail s'est porté sur la compréhension du problème et la passerelle entre les deux domaines.

Ce premier simulateur utilise l'IDM comme passerelle, le procédé est détaillé dans le paragraphe suivant.

4. L'IDM, passerelle entre IHM et planification

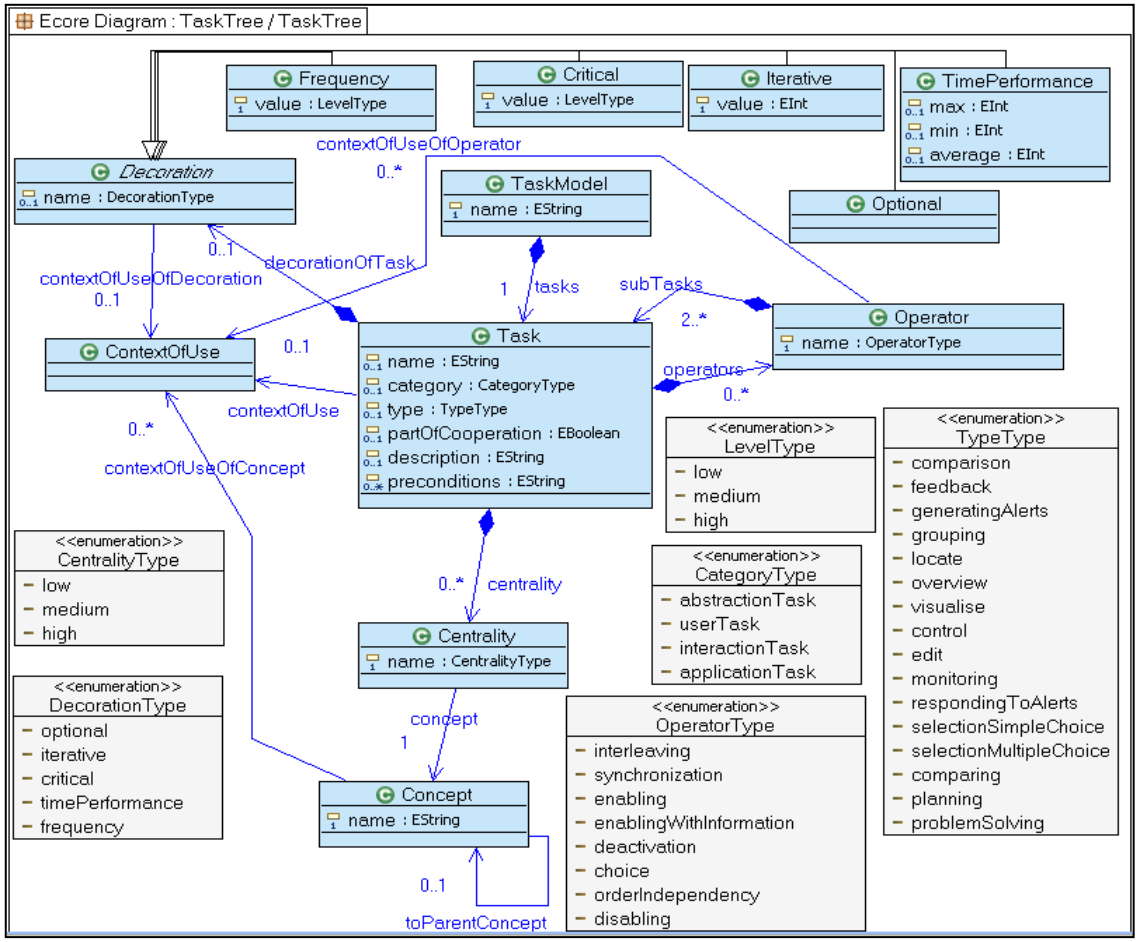


Fig. 6 - Notre métamodèle de tâches pour une prise en compte plus poussée du contexte d'usage.

Les transformations entre IHM et planification sont fondées sur des métamodèles explicites : une (méta)modélisation des tâches utilisateur (Sottet *et al.*, 2006) (Guilio & Paterno, 2004), du contexte d'usage et de la planification. Nous présentons les métamodèles de tâches et de planification utilisés. Ceci permet la mise en correspondance des deux domaines par des règles de transformations IDM.

4.1. Le métamodèle de tâches

Tenir compte du contexte d'usage étant un point clé du processus de composition, notre métamodèle de tâches (Fig. 6) permet d'estampiller toute information (les tâches utilisateur ; les opérateurs logiques ou temporels entre tâches ; les décorations de tâches telles que la fréquence ou la criticité) du contexte d'usage dans lequel elle fait sens.

4.2. Le métamodèle de la planification

Le métamodèle de la planification (Fig. 7) est générique, donc détaché des préoccupations liées à une grammaire quelconque. Il fait apparaitre les concepts fondamentaux de la planification (problème, domaine, etc.)

Le paragraphe suivant expose les correspondances et les distinctions entre IHM et planification.

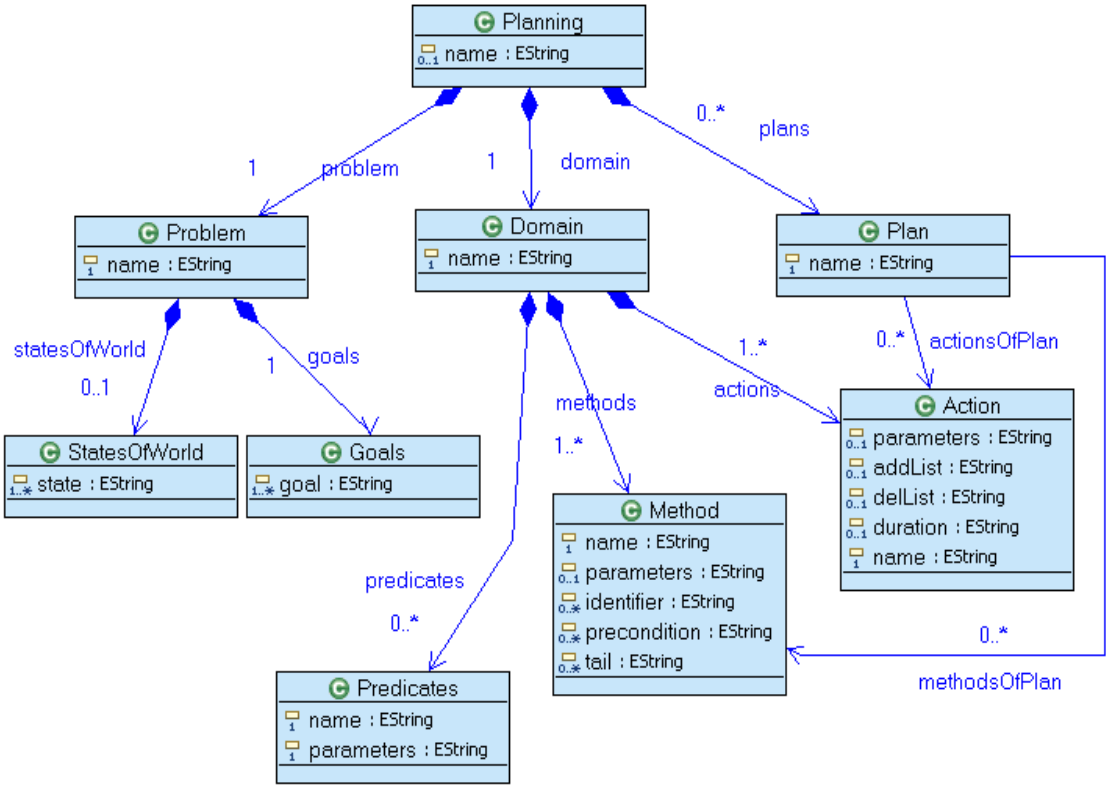


Fig. 7 -Métamodèle générique de la planification

5. Correspondances et distinctions entre IHM et Planification

Dans un premier temps, ce paragraphe aborde les apports possibles de la planification à la composition d'IHM et inversement. Dans un second temps, il met en évidence les différences et les difficultés engendrées par ce rapprochement.

5.1. Correspondances

La planification permet de résoudre un but à atteindre. Elle manipule une description de l'état du monde et un ensemble d'actions (opérateurs) possibles. La résolution du but s'exprime par un ensemble de plans décrits sous la forme d'une suite d'actions. Dans notre étude, l'objectif utilisateur doit être résolu dans un contexte d'usage et un ensemble de services variable. Le système doit donc fournir à l'utilisateur un choix de *n* IHM lui permettant de résoudre son but.

Mettre en correspondance les deux domaines équivaut à obtenir une expressivité équivalente entre les termes suivants d'IHM et de planification :

- l'objectif utilisateur et le but à atteindre : en planification, le but à atteindre équivaut à un état du monde à atteindre. L'objectif utilisateur doit donc être exprimé de cette manière ;
- le contexte d'usage et l'état du monde : la modélisation du contexte d'usage doit être exprimable, de la même manière que l'état du monde, par des prédicats pouvant posséder des paramètres ;
- un ensemble de services et un ensemble d'actions : un modèle de tâches représentant un service doit correspondre à une ou plusieurs actions. Une action est composée de paramètres, de pré-conditions qui doivent être vraies pour pouvoir effectuer l'action, et de post-conditions qui modifient l'état du monde si l'action est faisable ;
- un ensemble d'IHM et un ensemble de plans, en conséquence, une IHM et un plan : une IHM est représentée par un modèle de tâches. Un modèle de tâches est exprimé par une suite d'actions qui constituent un plan. Il en découle que représenter une IHM par un plan est cohérent avec les points précédents.

Cependant, en réalisant ces correspondances, des différences apparaissent.

5.2. Distinctions

La correspondance entre l'IHM et domaine de planification est néanmoins loin d'être immédiate :

1) Divergences terminologiques : la notion d'opérateur, par exemple, est un faux ami. En IHM, il dénote les relations logiques ou temporelles entre tâches. En planification, il s'apparente à une tâche (par exemple, allumer(X) où X est une plate-forme) ;
2) Divergences sémantiques : le « *ou* » en IHM représente un choix pour l'utilisateur à proposer dans l'interface finale tandis que le « *ou* » en planification réalise ce choix en ne sélectionnant, dans un plan solution, que l'une ou l'autre de ses alternatives ;
3) Préoccupations différentes : dans son calcul, le planificateur connaît les tâches utilisateur abstraites mais ne les fournit pas en sortie. Il restitue une séquence d'actions physiques (c'est-à-dire sans tâches abstraites)à partir de laquelle l'IHM devra recalculer les tâches abstraites. Or ces tâches abstraites sont utiles dans la construction d'IHM concrètes et finales ;
4) L'opérateur « *entrelacement* » qui permet à l'utilisateur de réaliser les mêmes tâches en même temps n'a pas d'équivalence évidente en planification ;

5) La décoration « *itération* » qui permet à une tâche d'être réalisée par l'utilisateur autant de fois qu'il le souhaite, pose un problème évident de récursivité infinie dans la construction du plan solution. Un planificateur ne trouvera pas de solution à cause de cette récursivité ;
6) Les décorations de tâches « *critique* », « *fréquente* » et « *optionnelle* » ne sont actuellement pas exprimables dans un domaine de planification.

Ces correspondances et différences entre domaines sont discutées dans le paragraphe suivant pour mettre en perspective les futurs travaux à réaliser.

5.3. Discussion

Dans une problématique de contexte d'usage et de besoins variables et imprévisibles, la planification propose des atouts indéniables : correspondance avec les entrées nécessaires en IHM (objectif utilisateur, contexte d'usage, actions) et les sorties en IHM (IHM permettant à l'utilisateur de réaliser son but) respectives, outils de planification existants qui permettent de calculer la procédure pour réaliser un objectif selon un état du monde.

Cependant, la mise en correspondance est moins immédiate qu'il n'y paraît et certaines différences d'expressivité entrainent des difficultés à résoudre. Celles-ci identifiées, cela permet d'orienter notre étude et d'envisager la modification des formalismes ou des algorithmes de planification pour que, par exemple, le planificateur soit capable de fournir un arbre possédant des actions abstraites plutôt qu'une simple suite d'actions. Cette problématique de correspondance permet de mettre en évidence aussi la différence fondamentale entre ces domaines : en IHM, un modèle de tâches peut exprimer un choix incombant à l'utilisateur tandis que la planification fournit une procédure pour réaliser un but sans intervention extérieure. Elle effectue les choix pour l'utilisateur.

6. Conclusion et perspectives

L'article propose un parcours en largueur de notre approche et du premier prototype pour la composition dynamique de systèmes interactifs. Nous combinons deux domaines, l'IHM et la planification. L'expérience montre que l'effort de mise en correspondance est loin d'être stérile. Il implique une meilleure compréhension en termes de sémantique et de complétude de chaque domaine et permet de mettre en relief les difficultés d'une telle problématique. En perspective, nous ciblons un travail visant à adapter les descriptions et les algorithmes de planification pour la composition automatique d'IHM et la prise en compte des contraintes ergonomiques : il ne s'agit pas simplement de trouver un plan solution, mais un plan réalisant les objectifs utilisateur et conduisant à une IHM utilisable. MAPPING v2 fera l'objet d'évaluations pour comprendre le degré d'observabilité et de contrôle à donner à l'utilisateur dans ce processus de construction.

7. Remerciements

Nous remercions la Région Rhône-Alpes, via le cluster ISLE, projet PRESENCE pour le financement de ce travail.

Références

BALBO S. (1994) Evaluation ergonomique des interfaces utilisateur : un pas vers l'automatisation,Thèse de doctorat Informatique préparée au Laboratoire de Génie Informatique (IMAG), Université Joseph Fourier 287 pages.

BELAUNDE BM. & AL., *MDA Guide Version 1.0.1*, Object Management Group, Inc. (June 12, 2003), http://www.omg.org/docs/omg/03-06-01.pdf.

COUTAZ J. (2001). Architectural Design for User Interfaces ; The Encyclopedia of Software Engineering, J. Marciniak Ed., Wiley & Sons Publ., second edition.

FLORINS M. (2006). Méthod for Designing Multiplatform Graphical User Interfaces, thèse de doctorat, université catholique de Louvain.

GUILIO M. & PATERNO F. & SANTORO C. (2004). Design and Development of Multidevice User Interfaces through Multiple Logical Description », *IEEE Transactions on Software Engineering*, vol. 20, n° 8, p. 507-520.

ILGHAMI O. & NAU DS. (2003). A general approach to synthesize problem-specific planners, Technical report CS-TR-4597, UMIACS-TR-2004-40, University of Maryland, October 2003.

LIU H. & SINGH P. (2004). Focusing on ConceptNet's natural language knowledge representation, Commonsense Reasoning in and over Natural Language Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information & Engineering Systems (Wellington, New Zealand, September 22 – 24, 2004) KES'2004, Lecture Notes in Artificial Intelligence.

MACKENZIE M. & LASKEY K. & MCCABE F. & BROWN P. & METZ R. (2006). Reference Model for Service Oriented Architecture 1.0, Technical Report n° wd-soa-rm-cd1, OASIS, February, 2006.

MYERS B. (1993). State of the Art in User Interface Software Tools. In Hartson, H. and Hix, D. (Eds.) *Advances in Human-Computer Interaction*.Ablex, pp. 110-150. Reprinted in: R.Baecker, J.Grudin, W.Buxton, and S. Greenberg, eds. *Readings in Human-Computer Interaction: Toward the Year 2000*. Second Edition. San Francisco: Morgan Kaufmann Publishers, Inc., 1995. pp. 344-356.

NAU D. & GHALLAB M. & TRAVERSO P. (2004). Automated Planning: Theory and Practice, Morgan Kaufmann Publishers Inc.

SCAPIN D. & BASTIEN JM. (1997). Ergonomic Criteria for evaluating the ergonomic quality of interactive systems, Behaviour & Information Technology, 6(4-5), 1997, pp 220-231.

SOTTET JS. & CALVARY G. & FAVRE JM. (2006). Towards mapping and model transformation for consistency of Plastic User Interfaces, *Computer Human Interaction*, Workshop on The Many Faces of Consistency in Cross-platform Design.