Résolution exacte et approchée de problèmes de décision markoviens formules en logique propositionnelle.

Boris Lesner, Bruno Zanuttini

Résolution exacte et approchée de problèmes de décision markoviens formulés en logique propositionnelle

Boris Lesner Bruno Zanuttini *

Résumé

In this paper we present a method for solving factored Markov Decision Processes described using propositional logic. We show that the complexity of solving such problems directly depends on that of the logical operations used, such as satisfiability testing. In order to reduce this complexity, we propose a problem approximation method into classes of formulas which are tractable for these operations. Using such approximations, we experimentally measure the capability of various classes of formulas to represent the structure of problems as well as the benefits in terms of resolution time.

Nous présentons une méthode factorisée de résolution de Processus de Décision Markoviens formulés en logique propositionnelle. Nous montrons que la complexité de la résolution de tels problèmes dépend directement de celles des opérations logiques mises en œuvre telles que le test de satisfaisabilité. Afin de réduire cette complexité, nous proposons une méthode d'approximation des problèmes dans des classes de formules permettant de réaliser de manière efficace ces opérations. En utilisant ces approximations, nous mesurons expérimentalement la capacité de différentes classes de formules à représenter la structure des problèmes, ainsi que les gains en temps de résolution.

1 Introduction

Les processus de décision markoviens (PDM) constituent une formalisation centrale pour l'étude de problèmes de planification sous incertitude. Sous leur forme classique, des méthodes de résolution telles que *Value Iteration* [2] requièrent une énumération de l'ensemble des états possibles du système. Cette contrainte limite leur application à des problèmes de taille relativement petite.

*Laboratoire GREYC (CNRS UMR 6072), Campus Côte de Nacre, boulevard du Maréchal Juin, BP 5186 – 14032 Caen CEDEX, boris.lesner@info.unicaen.fr, bruno.zanuttini@info.unicaen.fr

La dernière décennie a vu l'apparition de méthodes *structurées*, qui tirent parti des régularités et de la structure inhérentes aux problèmes concrets. Dans la littérature on parle alors de PDM factorisés (ou plus couramment fMDP). Ces techniques ont alors permis d'atteindre des problèmes aux espaces d'états considérablement plus vastes.

Dans le cas de la résolution exacte de problèmes factorisés, des techniques *d'agrégation* sont le plus souvent utilisées. Il s'agit de regrouper des états identiques du point de vue de la décision en un *macro-état* qui sera traité comme un état unique au cours de la résolution du problème. La représentation de ces ensembles d'états peut prendre différentes formes, en particulier des arbres de décision pour l'algorithme SVI [3] ou des diagrammes de décision algébriques pour SPUDD [7]. Dans tous les cas, l'espace d'états est considéré, de manière implicite, comme étant l'espace des valuations possibles d'un ensemble de variables booléennes.

Pour certains types de problèmes particuliers, des méthodes adaptées existent. D'une part, dans le cas où l'état initial est connu, les algorithmes SLAO* [**?**] et sRTDP [**?**] exploitent à la fois les représentations compactes et le fait que tout l'espace d'états n'est pas nécessairement atteignable. Ces deux algorithmes tirent aussi parti d'une heurisitique, représentée sous la forme d'une solution approchée du problème afin d'accéler la résolution. D'autre part, les méthodes approximatives bénéficient elles aussi des représentations symboliques, comme APRICODD [**?**] qui est une version approchée de SPUDD ou bien encore l'approche de [**?**] qui représente approximativement la fonction de valeur par combinaison linéaire de fonctions de base (elles-même représentées de manière factorisée).

La méthode présentée ici utilise la logique propositionnelle pour représenter des ensembles d'états. La motivation principale repose sur le fait que, typiquement, pour une action donnée, seule une petite partie des variables décrivant l'environnement est pertinente pour décrire cette action ; il en va de même pour représenter les fonctions d'utilité.

Notre seconde contribution est une méthode d'approximation des problèmes. Nous proposons de remplacer le problème original par un problème utilisant des classes de formules pour lesquelles la résolution est efficace. Les fonctions de valeur obtenues sur ces approximations fournissent un encadrement de la fonction de valeur réelle. Cette méthode peut être utilisée pour des approximations successives donnant un comportement « anytime » à la résolution.

Enfin, nous évaluons expérimentalement notre méthode sur des problèmes factorisés générés aléatoirement. Ces expériences montrent que l'utilisation des opérations sur des classes de formules « efficaces » est plus coûteuse que la représentation générique en diagrammes de décision binaires utilisée par notre solveur. Malgré tout, en utilisant cette représentation, nous montrons que l'approximation dans la classe des formules de Horn accélère significativement la résolution sans sacrifier la qualité des solutions.

Dans un premier temps sont abordés les processus de décision markoviens (section 2). Ensuite, après quelques préliminaires de logique propositionnelle (section 3.1), nous abordons la représentation (section 3.2) et notre algorithme

de résolution de tels processus formulés en logique (section 4). Dans la section 5 nous présentons le principe de notre méthode d'approximation, pour ensuite donner un exemple sur une classe de formules spécifique (section 6). Vient ensuite la description de l'utilisation « anytime » des approximations (section 7). Nous terminons sur la présentation de nos expériences à la section 8.

2 Processus de décision markoviens

Un PDM permet de modéliser un problème de planification sous incertitude. Il est muni d'une fonction de récompense représentant l'utilité de ses actions. Contrairement aux outils de planification « classiques », la résolution d'un PDM ne produit pas de plan (une séquence d'actions à réaliser menant à un but) mais une *politique* : une fonction qui associe à chaque état la *meilleure* action à exécuter. Cette politique maximise l'espérance de l'utilité reçue lors de son exécution.

Un PDM est un quadruplet $(\mathcal{S}, \mathcal{A}, T, R)$ qui consiste en un ensemble fini d'états $\mathcal{S}$ et d'actions $\mathcal{A}$. Les actions modélisent des transitions entre les états et sont définies par une fonction $T(s, a, s')$ qui donne la probabilité de passer de l'état s à s' en effectuant l'action a. Une fonction de récompense $R : \mathcal{S} \to \mathbb{R}$ affecte une valeur à chaque état, afin de différencier les états « favorables » de ceux « défavorables ».

Le but est de trouver une politique, c'est-à-dire une fonction $\pi : \mathcal{S} \to \mathcal{A}$ qui associe à chaque état une action maximisant l'espérance des récompenses futures. La politique peut être calculée à horizon fini (l'agent ne doit exécuter qu'un nombre fixé d'actions) ou infini (l'agent agit indéfiniment). L'espérance de récompense peut être pondérée par un *taux d'amortissement* $0 \leq \gamma < 1$. La valeur espérée $V_\pi(s)$ d'une politique π dans un état donné s satisfait l'équation suivante [1] :

$$V_\pi(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_\pi(s') \tag{1}$$

L'équation 1 exprime le fait que la valeur d'un état s selon une politique π est égale à la somme de la récompense immédiate pour s et de l'espérance de récompense qui est la somme des valeurs de chaque état s' accessible depuis s, chaque valeur $V(s')$ étant pondérée par la probabilité de passer de s à s' en exécutant l'action $\pi(s)$. Le coefficient γ représente l'importance donnée aux valeurs qui peuvent être obtenues dans le futur.

Une politique π est dite *optimale* si pour toute politique π' et tout état $s \in \mathcal{S}$, $V_\pi(s) \geq V_{\pi'}(s)$. On note V^* la *fonction de valeur optimale* qui est la fonction de valeur de toute politique optimale.

La méthode de résolution *Value Iteration* [2] calcule une fonction de valeur V^n à n étapes restantes et converge par itérations sucessives vers la fonction de

1. Nous nous limitions ici au cas où seul l'état courant détermine la récompense, mais nos résultats se généralisent à d'autres formes de fonctions de récompense.

valeur optimale. En prenant $V^0(s) = R(s)$ pour tout $s \in \mathcal{S}$, nous avons :

$$V^{n+1}(s) = R(s) + \max_{a \in \mathcal{A}} \left\{ \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V^n(s') \right\}$$

Pour un horizon n et une action a, nous notons $Q_a^n(s)$ la *Q-valeur* de l'état s, c'est-à-dire l'espérance de récompense en exécutant l'action a dans l'état s :

$$Q_a^n(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V^{n-1}(s')$$

La valeur optimale à l'horizon n peut donc s'exprimer comme celle de la Q-valeur maximale : $V^n(s) = \max_{a \in \mathcal{A}} \{Q_a^n(s)\}$. Une politique associée à l'itération n vérifie alors $\pi^n(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \{Q_a^n(s)\}$.

Dans le cas de la résolution à horizon infini, une valeur de γ strictement inférieure à 1 permet de garantir la convergence de la fonction de valeur après un nombre fini d'itérations. Dans ce cas, on stoppe l'algorithme quand la différence de valeur entre V^{n+1} et V^n est inférieure à une certaine valeur ϵ pour tout état. La politique déduite de cette fonction de valeur est alors dite *ϵ-optimale*.

3 PDM et logique propositionnelle

3.1 Notions de logique propositionnelle

Les expressions de la logique sont les *formules*, ici représentées par des lettres grecques (le plus souvent ϕ et ψ). Toute formule ϕ porte sur un ensemble de *variables propositionnelles* noté $\mathrm{Var}(\phi)$; ces variables peuvent prendre la valeur 1 (VRAI) ou 0 (FAUX). Les variables sont connectées par les opérateurs $\wedge$ (conjonction), $\vee$ (disjonction), $\neg$ (négation), $\rightarrow$ (implication). Ces opérateurs sont régis par la sémantique standard de la logique propositionnelle.

Un *littéral positif* est une formule constituée d'une seule variable, tandis qu'un *littéral négatif* est la négation d'une variable. On dit que les littéraux sont *formés* sur une variable, ainsi x et $\neg x$ sont les littéraux formés sur x. Pour un littéral ℓ on note $\overline{\ell}$ le littéral *complémentaire* de ℓ ; ainsi, si $\ell = \neg x$ nous avons $\overline{\ell} = x$.

Une *affectation* d'un ensemble fini de variables V est une application de V dans $\{0, 1\}$. L'ensemble $\{0, 1\}^V$ des affectations de V est noté 2^V pour plus de concision. L'affectation à 1 d'une variable x est notée simplement x tandis que son affectation à 0 est notée $\overline{x}$. Par exemple, pour $V = \{x, y, z\}$, $x\overline{y}z$ est une affectation de V.

Les *modèles* d'une formule propositionnelle ϕ sont toutes les affectations m de $\mathrm{Var}(\phi)$ qui satisfont ϕ (noté $m \models \phi$). Cet ensemble de modèles est noté $\mathcal{M}(\phi)$. Par exemple, pour $\phi = x \vee y$, nous avons $\mathcal{M}(\phi) = \{xy, \overline{x}y, x\overline{y}\}$.

Un *terme* (resp. une *clause*) est une conjonction (resp. une disjonction) de littéraux. Afin de clarifier les notations, nous considèrerons parfois un terme comme l'ensemble de ses littéraux.

Le *test de satisfaisabilité* (ci-après noté test SAT) consiste à déterminer s'il existe au moins une affectation satisfaisant une formule. Dans le cas général ce test est NP-complet, mais nous verrons par la suite qu'il existe des classes de formules pour lesquelles il est polynomial.

Une opération centrale pour notre travail est *l'oubli de variables*, dont les propriétés sont discutées en détail dans [8]. Intuitivement, oublier un ensemble de variables V dans une formule ϕ consiste à construire une formule $Oubli(V, \phi)$ dont les modèles sont une *projection* de $\mathcal{M}(\phi)$ sur l'ensemble de variables $\mathrm{Var}(\phi) \setminus V$.

Définition 1 (oubli de variables) *L'oubli d'un ensemble de variables dans une formule ϕ est tel que :*
- $Oubli(\emptyset, \phi) = \phi$
- $Oubli(\{x\}, \phi) = \phi_{x \leftarrow 0} \vee \phi_{x \leftarrow 1}$
- $Oubli(\{x\} \cup V, \phi) = Oubli(V, Oubli(\{x\}, \phi))$

Exemple 1 *Soit $\phi = (\neg x \vee y) \wedge (z \vee \neg y)$. On a $\mathcal{M}(\phi) = \{\overline{xyz}, \overline{xy}z, \overline{x}yz, xyz\}$. En conséquence, $\mathcal{M}(Oubli(\{y\}, \phi)) = \{\overline{x}\overline{z}, \overline{x}z, xz\}$. Donc $\neg x \vee z$ est un oubli de $\{y\}$ dans ϕ.*

3.2 Représentation d'un processus de décision markovien

La fonction de récompense est représentée de manière factorisée par un ensemble $R = \{(\phi_1, r_1), \ldots, (\phi_k, r_k)\}$ où chaque ϕ_i est une formule propositionnelle et $r_i \in \mathbb{R}$. La récompense $R(s)$ alors associée à un état s est la somme des r_i tels que $s \models \phi_i$.

Chaque action a est représentée en utilisant des Opérateurs STRIPS Probabilistes (PSO), [**?**, 4]. À une action $a = (\mathcal{C}_a, \mathcal{E}_a)$ sont associés deux ensembles $\mathcal{C}_a = \{c_1^a, \ldots, c_{n_a}^a\}$ et $\mathcal{E}_a = \{E_1^a, \ldots, E_{n_a}^a\}$; $\mathcal{C}_a$ est un ensemble de *conditions* : des formules mutuellement inconsistantes, et $\mathcal{E}_a$ associe à chaque condition c_i^a un ensemble E_i^a *d'effets* étant chacun un *terme*. Un effet représente les modifications apportées à un état si on y exécute l'action correspondante, les variables non mentionnées dans l'effet ne changeant pas. L'ensemble des variables propositionnelles utilisées pour représenter le problème est noté $\mathcal{D}$. Ces notations sont illustrées sur le problème donné dans l'exemple 3. Le calcul de l'état résultant de l'application d'un effet à un état est donné par la définition 2.

Définition 2 (application d'un effet à un état) *Appliquer un effet e à un état s consiste à retirer de s tous les littéraux dont la variable est modifiée par e, puis d'en faire l'union avec e. Cette application est notée $ApplS$:*

$$ApplS(e, s) = \left(s \setminus \left\{\ell \mid \bar{\ell} \in e\right\}\right) \cup e$$

Exemple 2 *Pour un état $s = xy\overline{z}t$ et un effet $e = \neg x \wedge z \wedge t$ nous avons : $ApplS(e, s) = \overline{x}yzt$.*

Le non-déterminisme est représenté par l'ajout à chaque effet $e_{i,j}^a \in E_i^a$ d'une probabilité $p_{i,j}^a$ telle que $\sum\{p_{i,j}^a \mid e_{i,j}^a \in E_i^a\} = 1$. Pour déterminer les

états atteignables depuis un état s avec une action a on cherche une condition c_i^a telle que $s \models c_i^a$, s'il n'existe pas de telle condition alors l'action a n'est pas possible dans l'état s. Notons $\mathit{Eff}(a, s)$ l'ensemble des effets applicables pour une action a et un état s :

$$\mathit{Eff}(a, s) = \begin{cases} E_i^a & \text{si } \exists c_i^a,\ s \models c_i^a \\ \emptyset & \text{sinon} \end{cases}$$

Il est possible de déterminer le PDM correspondant de la manière suivante :
- $\mathcal{S} = 2^{\mathcal{D}}$.
- $T(s, a, s') = p_{i,j}^a$ si $\exists e_{i,j}^a \in \mathit{Eff}(a, s)$ tq. $s' = \mathit{ApplS}(e_{i,j}^a, s)$.
- $T(s, a, s') = 0$ sinon.
- $R(s) = \sum\{r \mid (\phi, r) \in R, s \models \phi\}$

Exemple 3 *Le tableau 3.2 donne la représentation tabulaire des actions d'une variante du problème* Coffee *présenté dans [4]. Ce problème met en scène un robot devant acheter puis livrer du café à un utilisateur et est décrit par 6 variables : Office : le robot au bureau sinon au café ; RhC : le robot possède du café ; UhC : l'utilisateur possède du café ; Rain : il pleut à l'extérieur ; Wet : le robot est mouillé ; Umb : le robot a un parapluie. Il dispose de 4 actions : Move : se déplacer, GetU : prendre le parapluie ; BuyC : acheter du café et DelC : donner le café. La fonction de récompense est* $R = \{(\mathit{UhC}, 0.8), (\neg \mathit{Wet}, 0.2)\}$ *: ses objectifs sont de livrer du café à l'utilisateur et de ne pas être mouillé lorsqu'il se déplace.*

Pour expliciter les notations, nous avons, par exemple pour l'action DelC :
- $\mathcal{C}_{DelC} = \{C_1^{DelC} = \mathit{Office} \wedge \mathit{RhC}, C_2^{DelC} = \neg \mathit{Office} \wedge \mathit{RhC}\}$
- $E_1^{DelC} = \{e_{1,1}^{DelC} = \neg \mathit{RhC} \wedge \mathit{UhC}, e_{1,2}^{DelC} = \neg \mathit{RhC}, e_{1,3}^{DelC} = \emptyset\}$,
- $E_2^{DelC} = \{e_{2,1}^{DelC} = \neg \mathit{RhC}, e_{2,2}^{DelC} = \emptyset\}$
- $\mathcal{E}_{DelC} = \{E_1^{DelC}, E_2^{DelC}\}$
- $\mathit{Eff}(DelC, \overline{\mathit{Office}}\ \mathit{RhC}\ \overline{\mathit{UhC}}\ \mathit{Rain}\ \mathit{Wet}\ \mathit{Umb}) = \{\neg \mathit{RhC}, \emptyset\}$

Cette représentation compacte des actions par les Opérateurs STRIPS Probabilistes est une des différentes possibles pour les PDM factorisés. Elle tire parti du fait que les actions d'un problème n'ont en général que peu d'effets différents, et ne portent que sur un nombre restreint de variables. Une autre représentation très courante dans la littérature est celle des réseaux bayésiens dynamiques [**?**, **?**] qui tire parti cette fois de la faible corrélation entre les variables du problème. Pour une comparaison détaillée des deux formalismes de représentation compacte, nous référons le lecteur à [**?**].

4 Résolution factorisée

Nous proposons désormais notre algorithme de résolution de problèmes représentés comme ci-dessus. Dans un premier temps nous abordons son fonctionnement. Ensuite, nous terminons par un aperçu de sa complexité.

Action	Condition	Effet	Prob.
Move	$Office \wedge Rain \wedge \neg Umb$	$\neg Office \wedge Wet$	0.8
		$\neg Office$	0.1
		$\emptyset$	0.1
	$Office \wedge (\neg Rain \vee Umb)$	$\neg Office$	0.9
		$\emptyset$	0.1
	$\neg Office \wedge Rain \wedge \neg Umb$	$Office \wedge Wet$	0.8
		$Office$	0.1
		$\emptyset$	0.1
	$\neg Office \wedge (\neg Rain \vee Umb)$	$Office$	0.9
		$\emptyset$	0.1
BuyC	$Office$	RhC	0.8
		$\emptyset$	0.2
GetU	$Office \wedge \neg Umb$	Umb	0.9
		$\emptyset$	0.1
DelC	$Office \wedge RhC$	$\neg RhC \wedge UhC$	0.8
		$\neg RhC$	0.1
		$\emptyset$	0.1
	$\neg Office \wedge RhC$	$\neg RhC$	0.8
		$\emptyset$	0.2

TABLE 1 – Exemple de représentation STRIPS Probabiliste

4.1 Algorithme

La résolution de PDM ainsi formulés se base sur l'algorithme *Value Iteration*, dont le but est de calculer une fonction de valeur optimale. De manière factorisée, une fonction de valeur V est représentée sous la forme d'une *partition* : un ensemble de couples (formule, valeur) où les formules sont mutuellement inconsistantes et l'union de leurs ensembles de modèles couvre $2^{\mathcal{D}}$. Ainsi, pour tout état $s \in 2^{\mathcal{D}}$ il existe exactement un couple $(\phi, v) \in V$ tel que $s \models \phi$ et v est sa valeur associée selon V. Par abus de notation nous noterons $V(\phi)$ les éléments de V qui représentent les valeurs possibles selon V pour les états représentés par ϕ :

$$V(\phi) = \{(\psi, v) \in V \mid \phi \wedge \psi \text{ est satisfaisable}\}$$

L'algorithme de résolution se déroule en trois étapes : initialiser une première partition de valeur grossière à partir de la récompense ; raffiner itérativement cette partition et ses valeurs comme le fait *Value Iteration* ; une fois que les valeurs de la partition ont convergé, en déduire la politique associée.

La première étape constitue à créer la partition V^0. Elle est calculée de manière à distinguer toutes les situations de récompense initiale. Notons la fonction de récompense $R = \{(\psi_1, r_1), \ldots, (\psi_{|R|}, r_{|R|})\}$ et, pour toute formule ψ, $\psi^0 \equiv \neg\psi$ et $\psi^1 \equiv \psi$. Alors

$$V^0 = \left\{ \left(\phi = \psi_1^{\epsilon_1} \wedge \cdots \wedge \psi_{|R|}^{\epsilon_{|R|}} \ , \ \epsilon_1 r_1 + \cdots + \epsilon_{|R|} r_{|R|} \right) \mid \forall i, \epsilon_i \in \{0,1\}, \ \phi \text{ est sat.} \right\}$$

Exemple 4 *Soit* $R = \{(x, 1), (\neg x \vee y, 3)\}$ *une fonction de récompense factorisée. Alors* $V^0 = \{(x \wedge y, 4), (x \wedge \neg y, 1), (\neg x, 3)\}$

La partition V^0 est la plus grossière possible permettant de distinguer les états ayant la même récompense immédiate, aux coïncidences près. À chaque itération de la résolution, ce seront les formules de V^0 qui seront raffinées selon la partition de valeur de l'itération précédente pour former la nouvelle partition. Le raffinement se déroule comme suit : pour chaque formule ϕ de V^0, calculer les ensembles d'états résultant de l'application de tous les effets de chaque action sur les états représentés par ϕ. Si, pour les états d'un ensemble ainsi calculé, les valeurs diffèrent selon la fonction de valeur de l'itération précédente, alors cet ensemble est décomposé en autant de sous-ensembles disjoints qu'il y a de valeurs possibles. Ces opérations sont réalisables de manière factorisée, en maintenant la représentation des états sous la forme de formules dont les modèles ne sont pas énumérés.

La formule (l'ensemble d'états) résultant de l'application d'un effet e sur une formule ϕ est notée $\mathit{ApplF}(e, \phi)$ et est telle que :

$$\mathcal{M}(\mathit{ApplF}(e, \phi)) = \{\mathit{ApplS}(e, s) \mid s \in \mathcal{M}(\phi)\}$$

Lemme 1 *Pour toute conjonction* $\phi_1 \wedge \phi_2$*, l'application d'un effet* e *implique la conjonction des applications :*

$$\mathit{ApplF}(e, \phi_1 \wedge \phi_2) \models \mathit{ApplF}(e, \phi_1) \wedge \mathit{ApplF}(e, \phi_2)$$

Démonstration. Par définition de l'implication logique, il faut que les modèles de la partie de gauche soient inclus dans ceux de la partie de droite. Prenons $s \in \mathcal{M}(\mathit{ApplF}(e, \phi_1 \wedge \phi_2))$, s est le résultat de l'application de s à un modèle commun à ϕ_1 et ϕ_2 ; c'est-à-dire $\exists m,\ m \in (\mathcal{M}(\phi_1) \cap \mathcal{M}(\phi_2))$ tel que $s = \mathit{ApplS}(e, m)$.

En appliquant maintenant l'effet e directement aux modèles de ϕ_1 et de ϕ_2 on obtient : $s \in (\{\mathit{ApplS}(e, m_1) \mid m_1 \in \mathcal{M}(\phi_1)\} \cap \{\mathit{ApplS}(e, m_2) \mid m_2 \in \mathcal{M}(\phi_2)\}))$. Ce qui signifie :

$$s \in \mathcal{M}(\mathit{ApplF}(e, \phi_1) \wedge \mathit{ApplF}(e, \phi_2))$$

□

Proposition 2 *L'application d'un effet peut être calculée via* l'oubli de variables *:*

$$\mathit{ApplF}(e, \phi) \equiv e \wedge \mathit{Oubli}(\mathrm{Var}(e), \phi)$$

Démonstration. Soit $\mathcal{L}(V) = \{v, \neg v \mid v \in V\}$ l'ensemble des littéraux pouvant être formés sur un ensemble de variables V. On a :

$$\begin{aligned}
\mathcal{M}(\mathit{ApplF}(e, \phi)) &= \{(m \setminus \mathcal{L}(\mathrm{Var}(e))) \cup e \mid m \in \mathcal{M}(\phi)\} \\
&= \{m \cup e \mid m \in \mathcal{M}(\mathit{Oubli}(\mathrm{Var}(e), \phi))\} \\
&= \mathcal{M}(e \wedge \mathit{Oubli}(\mathrm{Var}(e), \phi))
\end{aligned}$$

□

Raffiner une formule ϕ selon une partition de valeur V et une action a consiste à calculer un ensemble de formules $\phi \wedge \delta_1, \ldots, \phi \wedge \delta_k$ partitionnant ϕ telles que la Q-valeur selon a des états représentés par chaque $\phi \wedge \delta_i$ soit unique.

Lemme 3 *Soient une formule ϕ, un effet e et une partition de valeurs V. Alors, pour tout couple $(\psi_i, v_i) \in V(ApplF(e, \phi))$ on a :*

$$V(ApplF(e, Oubli(\mathrm{Var}(e), \psi_i \wedge e) \wedge \phi)) = \{(\psi_i, v_i)\}$$

Démonstration. Soit δ tel que $\delta = Oubli(\mathrm{Var}(e), \psi_i \wedge e)$. Alors d'après le lemme 1 nous avons $ApplF(e, \phi \wedge \delta) \models ApplF(e, \phi) \wedge ApplF(e, \delta)$. Or, $ApplF(e, \delta) \equiv \delta \wedge e$ car $\mathrm{Var}(\delta) \cap \mathrm{Var}(e) = \emptyset$ et $\delta \wedge e \equiv \psi_i \wedge e$ par définition de δ. Finalement, $ApplF(e, \phi) \wedge ApplF(e, \delta) \equiv ApplF(e, \phi) \wedge \delta \wedge e \equiv ApplF(e, \phi) \wedge \psi_i \wedge e \models \psi_i$, donc $ApplF(e, \phi \wedge \delta) \models \psi_i$ et ψ_i est alors la seule formule de V satisfaisable avec $ApplF(e, \phi \wedge \delta)$. □

À partir du lemme 3 nous pouvons construire l'algorithme 1 qui, étant données une formule ϕ, une partition de valeurs V et une condition d'action, partitionne ϕ comme ci-dessus. Il procède en appliquant récursivement chaque effet à ϕ. Si, après une application d'un effet e sur une formule φ, plusieurs valeurs sont possibles selon V, il calcule une formule de *distinction* δ_l pour chacune de ces valeurs.

Algorithme 1 : Raffiner(V : partition, ϕ : formule, c_i^a : condition)

si $\phi \wedge c_i^a$ *est NON SAT* **alors retourner** $\emptyset$
$I \leftarrow \{(\phi \wedge c_i^a, 0)\}$
pour chaque *effet* $e_{i,j}^a \in E_i^a$ **faire**
  $I' \leftarrow \emptyset$
  pour chaque $(\phi', v) \in I$ **faire**
    pour chaque $(\psi_l, v_l) \in V(ApplF(e_{i,j}^a, \phi'))$ **faire**
      $\delta_l \leftarrow Oubli(\mathrm{Var}(e_{i,j}^a), \psi_l \wedge e_{i,j}^a)$
      `/* Si la disjonction est disponible et/ou efficace pour la classe de formules considée, on peut procéder à une fusion (par disjonction) des formules ayant la même valeur */`
      $I' \leftarrow I' \cup \{(\phi' \wedge \delta_l,\ v + p_{i,j}^a \times v_l)\}$
  $I \leftarrow I'$
retourner I

Parfois l'algorithme 1 peut produire différentes formules avec des valeurs identiques. Dans ce cas, il est possible de fusionner de telles formules grâce à une disjonction pour réduire au maximum la taille de la partition résultat. Il faut cependant s'assurer que la disjonction est disponible dans la classe de formules considérée et si c'est le cas qu'elle est efficace. Il est en effet parfois plus raisonnable de laisser deux formules simples plutôt que d'utiliser leur disjonction qui peut être de taille sensiblement plus importante.

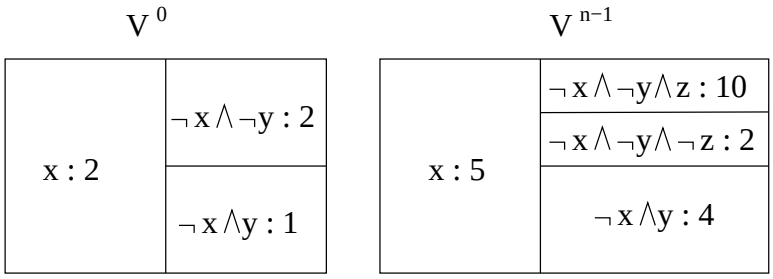

FIGURE 1 – Exemple de partitions de valeur

Exemple 5 (raffinement de partition par l'algorithme 1) *La figure 4.1 montre un exemple de partition initiale V^0 et une partition V^{n-1}. Supposons $c_i^a = x$, un unique effet $e = \neg x \wedge z$. Soit $\phi = x$. La formule $c_i^a \wedge \phi$ étant satisfaisable nous pouvons appliquer e : $ApplF(e, c_i^a \wedge \phi) = \neg x \wedge z$. On a $V^{n-1}(\neg x \wedge z) = \{(\neg x \wedge \neg y \wedge z, 10), (\neg x \wedge y, 4)\}$, il y a plus d'une valeur possible, il faut donc raffiner $c_i^a \wedge \phi$ en deux formules $c_i^a \wedge \phi \wedge \delta_1$ et $c_i^a \wedge \phi \wedge \delta_2$:*

- $\delta_1 = Oubli(\mathrm{Var}(e), \neg x \wedge \neg y \wedge z \wedge e) \equiv \neg y$ *: nous obtenons* $V^{n-1}(ApplF(e, c_i^a \wedge \phi \wedge \delta_1)) = \{(\neg x \wedge \neg y \wedge z, 10)\}$*, il y a bien une unique valeur pour ces états. La Q-valeur Q_a contiendra donc une formule $c_i^a \wedge \phi \wedge \delta_1$ munie de la valeur $2 + \gamma 10$.*
- $\delta_2 = Oubli(\mathrm{Var}(e), (\neg x \wedge y \wedge e) \equiv y$ *: nous obtenons* $V^{n-1}(ApplF(e, c_i^a \wedge \phi \wedge \delta_2)) = \{(\neg x \wedge y, 4)\}$*, Q_a contiendra $(c_i^a \wedge \phi \wedge \delta_2,\ 2 + \gamma 4)$.*

En appliquant pour chaque action l'algorithme 1 sur chacune des formules partitionnant V^0, nous obtenons une partition représentant la Q-valeur de chaque action. De manière similaire à *Value Iteration*, il faut maintenant maximiser les Q-valeurs, c'est à dire calculer une partition de valeurs maximales.

Pour le calcul d'une telle partition maximale nous proposons deux algorithmes différents. Le premier (algorithme 2) calcule une partition maximale en prenant les intersections de tous les éléments des Q-valeurs données, en gardant pour chacune d'elles la valeur maximale. Pour ce faire il utilise *seulement* la conjonction de formules et le test de satisfaisabilité. Cependant, il génère un grand nombre d'intersections candidates dont beaucoup se révèlent être vides.

Le second algorithme (algorithme 3) utilise à la fois la disjonction, la négation, et la conjonction sur les formules. Il choisit d'abord l'ensemble d'états ayant la Q-valeur maximale parmi toutes les partitions. Il choisit ensuite le second et n'en retient que la partie non couverte par le premier, et ainsi de suite, jusqu'à couvrir toute la partition demandée (paramètre Φ de l'algorithme 3).

L'algorithme 4 calcule donc la fonction de valeur optimale sous la forme d'une partition. Avant de raffiner V^0 pour calculer une Q-valeur, il initialise celle-ci en donnant une récompense de $-\infty$ pour les états où l'action courante n'est pas exécutable. Ainsi, cette action ne sera pas choisie lors de la maximisation. Après cette étape, les états où aucune action n'est possible ont une valeur de $-\infty$ qu'il

Algorithme 2 : Maximiser($\mathcal{Q}$: ensemble de Q-valeurs, Φ : couverture de $\mathcal{Q}$)

```
Prendre Q_{a_0}, la Q-valeur de l'action a_0 dans Q
V ← {(φ, v, a_0) | (φ, v) ∈ Q_{a_0}}
pour chaque action a_i ∈ A, i > 0 faire
    V' ← ∅
    Prendre Q_{a_i} la Q-valeur de a_i dans Q
    pour chaque (φ, v) ∈ Q_{a_i} faire
        pour chaque (φ', v', a) ∈ V faire
            ψ ← φ ∧ φ'
            si ψ est SAT alors
                si v > v' alors
                    V' ← V' ∪ {(ψ, v, a_i)}
                sinon
                    V' ← V' ∪ {(ψ, v', a)}
    V ← V'
π_a ← 0 ∀a ∈ A
M ← ∅
pour chaque (φ, v, a) ∈ V faire
    π_a ← π_a ∨ φ
    /* Note : pour éviter la disjonction, on peut garder la
       politique telle qu'elle est dans V.                  */
    M ← M ∪ {(φ, v)}
Π ← {π_a}_{a∈A}
retourner (M, Π)
```

faut remplacer par la récompense immédiate. Notons que pour plus d'efficacité nous pouvons maximiser seulement les raffinements de chaque formule ϕ_i de V^0 selon chaque action puisque les raffinements des $\phi_j, j \neq i$ sont nécessairement insatisfaisables avec ceux de ϕ_i.

4.2 Génération d'une politique

Une politique est simplement représentée par un ensemble de $|\mathcal{A}|$ formules, chacune associée à une action. L'action d'une telle formule est optimale pour tous les états la satisfaisant.

Une politique optimale pour les Q-valeurs courantes peut être calculée au cours de la maximisation, comme montré dans les algorithmes 2 et 3. À tout moment dans l'exécution de ces algorithmes, V est correcte (mais incomplète). On remarque en effet que seuls des éléments maximaux y sont insérés. Sachant que chaque élément inséré provient d'une unique action, il est possible de mettre directement à jour la formule de la politique de cette action.

Algorithme 3 : Maximiser($\mathcal{Q}$: ensemble de Q-valeurs, Φ : couverture de $\mathcal{Q}$)

$V \leftarrow \emptyset$
$\Psi \leftarrow 0$ `// rien n'est couvert`
$\pi_a \leftarrow 0 \; \forall a \in \mathcal{A}$
tant que $\Psi \not\equiv \Phi$ **faire**
  $(\phi, v)_a \leftarrow$`retirer-max` $(\mathcal{Q})$
  si $\phi \models \neg\Psi$ `//` ϕ `est totalement non couvert`
  alors
    $V \leftarrow V \cup \{(\phi, v)\}$
    $\Psi \leftarrow \Psi \vee \phi$
    $\pi_a \leftarrow \pi_a \vee \phi$
  sinon si $\phi \not\models \Psi$ `//` ϕ `est partiellement non couvert`
  alors
    $V \leftarrow V \cup \{(\neg\Psi \wedge \phi, v)\}$
    $\Psi \leftarrow \Psi \vee \phi$
    $\pi_a \leftarrow \pi_a \vee (\neg\Psi \wedge \phi)$
retourner $(V, \{\pi_a\}_{a \in \mathcal{A}})$

Algorithme 4 : Value Iteration sur un espace d'états factorisé

Initialiser V^0 avec la récompense
$n \leftarrow 0$
répéter
  $n \leftarrow n + 1$
  $V^n \leftarrow \emptyset$
  pour chaque $(\phi, r) \in V^0$ **faire**
    pour chaque $a \in \mathcal{A}$ **faire**
      $Q_{a,\phi} \leftarrow \{(\neg(c^a_1 \vee \cdots \vee c^a_{n_a}) \wedge \phi, -\infty)\}$
      pour chaque $c^a_i \in \mathcal{C}^a$ **faire**
        $Q_{a,\phi} \leftarrow Q_{a,\phi} \cup \{(\psi, r + \gamma \cdot v) \mid (\psi, v) \in$ `Raffiner`$(V^{n-1}, \phi, c^a_i)\}$
    $(V_\phi, \Pi_\phi) \leftarrow$ `Maximiser`$(\{Q_{a,\phi}\}_{a \in \mathcal{A}}, \phi)$
    Remplacer les $-\infty$ de V_ϕ par r
    $V^n \leftarrow V^n \cup V_\phi$
    $\Pi_a \leftarrow \Pi_a \vee \pi_a, \; \pi_a \in \Pi_\phi, \; \forall a \in \mathcal{A}$
jusqu'à V^n satisfaisante ou horizon atteint

4.3 Complexité

Nous cherchons à mesurer la complexité de notre algorithme en termes d'opérations logiques : test de satisfabilité, oubli de variables et conjonction.

Notons e, le nombre maximal d'effets par condition et $|V|$ la taille de la

partition de valeur courante. Pour l'algorithme 1, au plus $O(|V|^e)$ oublis de variables sont nécessaires. Sachant que vérifier les valeurs pour une formule donnée nécessite $O(|V|)$ tests de satisfaisabilité, il faudra dans le pire des cas $O(|V|^e)$ tests SAT. On peut montrer qu'il s'agit du nombre de raffinements à générer dans le pire des cas. L'exemple 6 montre un cas particulier où il faut générer un nombre exponentiel de distinctions.

Exemple 6 *Supposons que la partition courante V^{n-1} contienne :*

$$\neg x_1 \wedge x_2 \wedge x_3 \wedge \cdots \wedge x_k \wedge y_1 : 1 \quad \neg x_1 \wedge x_2 \wedge x_3 \wedge \cdots \wedge x_k \wedge \neg y_1 : 0$$
$$\cdots$$
$$x_1 \wedge x_2 \wedge x_3 \wedge \cdots \wedge \neg x_k \wedge y_k : 2^k \quad x_1 \wedge x_2 \wedge x_3 \wedge \cdots \wedge \neg x_k \wedge \neg y_k : 0$$

Pour chaque i, tous les x_j, $j \neq i$ sont positifs sauf x_i, et une puissance de 2 comme valeur selon que y_i est vrai ($V^{n-1} = 2^i$) ou faux ($V^{n-1} = 0$). On a donc $|V^{n-1}| = 2k + 1$ en comptant une dernière formule qui regroupe tous les autres états et de valeur quelconque.

Soit maintenant une action a avec une seule condition toujours vraie, et k effets : x_1, x_2, ..., x_k, chacun avec la même probabilité $1/k$. Supposons $(x_1 \wedge x_2 \wedge \cdots \wedge x_k, v) \in V^0$. Alors on peut voir que toute combinaison de y_i donne une valeur espérée différente : $Q_a(x_1 \wedge \cdots \wedge x_k \wedge y_1^{\epsilon_1} \wedge \cdots \wedge y_k^{\epsilon_k}) = v + \gamma \times 1/k \times (\sum \epsilon_i 2^i)$, $\forall \epsilon_i \in \{0, 1\}$.

Il faut donc bien 2^k distinctions, à comparer à $2k + 1$ pour $|V^{n-1}|$ et k pour e.

Malgré des complexités dans le pire des cas exponentielles, l'objectif de la résolution factorisée est de tirer parti du fait que e est typiquement petit par rapport au nombre de variables $|\mathcal{D}|$ et que $2^{|\mathcal{D}|}$ états sont à énumérer pour une résolution non factorisée.

L'algorithme 3 (maximiser les Q-valeurs) demande quant à lui $O(|Q||\mathcal{A}|)$ tests de satisfaisabilité. La version ne nécessitant que la conjonction (algorithme 2) requiert quant à elle $O(|Q|^{|\mathcal{A}|})$ tests SAT.

Le théorème suivant nous permettra de tirer parti de la faible complexité des opérations logiques [2] de certaines classes de formules en section 5.

Théorème 1 *Si les formules de V^0 et les conditions d'action sont dans une classe $\mathcal{C}$ stable pour la conjonction et l'oubli de variables, alors les formules de V^n sont dans $\mathcal{C}$ pour tout horizon n, lorsque l'algorithme 4 maximise avec l'algorithme 2.*

Démonstration. Par construction, l'algorithme exécute pour seules manipulations de formules la conjonction et l'oubli de variables. Si V^0 et les conditions sont dans une classe stable pour ces opérations, alors l'algorithme est stable pour cette classe. □

5 Approximation de problèmes

Nous avons vu dans la section précédente que la complexité de notre algorithme repose sur une grande quantité d'opérations logiques telles que le test de satisfaisabilité et l'oubli de variables. Dans le cas général, ces deux opérations sont difficiles, cependant, certaines classes de formules permettent de les

2. Nous ignorons pour le moment la formule $\neg(c_1^a \vee \cdots \vee c_{n_a}^a)$: cf. discussion en section 8.

réaliser en temps polynomial. Ces classes *efficaces* étant incomplètes, nous proposons donc une méthode *d'approximation* de problèmes factorisés dans de telles classes. Le théorème 1 nous garantit de rester dans la même classe si celle-ci est stable pour la conjonction et l'oubli de variables.

5.1 Encadrement de formules propositionnelles

Comme présenté dans [10], toute formule en logique propositionnelle peut être approchée dans des classes incomplètes telles que les formules de Horn, 2-CNF ou encore affines. L'approximation consiste à encadrer une formule initiale par deux formules, l'une étant logiquement plus forte et l'autre plus faible. Pour une formule ϕ exprimée dans un langage quelconque, elle peut être encadrée par deux formules ϕ_{lb} et ϕ_{ub} telles que $\phi_{lb} \models \phi \models \phi_{ub}$. Du point de vue des modèles (les états représentés par une formule) nous avons $\mathcal{M}(\phi_{lb}) \subseteq \mathcal{M}(\phi) \subseteq \mathcal{M}(\phi_{ub})$. La formule ϕ_{lb} (respectivement ϕ_{ub}) est un *minorant* (resp. *majorant*) de ϕ.

Nous proposons d'approximer des PDM factorisés par des PDM utilisant une classe de formules efficace pour notre algorithme de résolution, afin d'obtenir un encadrement facilement calculable de la valeur optimale du problème initial, dans l'esprit des travaux de [10] qui utilisent les majorants et minorants pour répondre à des requêtes d'implication et de satisfaisabilité.

Nous proposons donc d'encadrer un PDM factorisé quelconque P par deux problèmes approchés $P_\downarrow$ et $P_\uparrow$ dont les fonctions de valeurs optimales respectives ${V_\downarrow}^n$ et ${V_\uparrow}^n$ satisfont en tout état s et horizon n

$$ {V_\downarrow}^n(s) \leq V^n(s) \leq {V_\uparrow}^n(s) $$

Dans les sections suivantes nous montrons la contruction de $P_\downarrow$ et $P_\uparrow$ par approximation logique de la fonction de récompense et des actions de P.

5.2 Encadrement de la fonction de récompense

Dans ce cadre, pour un PDM P et sa fonction de récompense factorisée R, le problème approché $P_\downarrow$ est obtenu à partir de P en renforçant les formules de récompense positive et affaiblissant les négatives (et inversement pour $P_\uparrow$). Intuitivement la récompense de chaque état de $P_\downarrow$ est plus faible que dans P. Plus formellement, soit $P = (\mathcal{D}, \mathcal{A}, R)$ un PDM factorisé, $P_\downarrow = (\mathcal{D}, \mathcal{A}, R_\downarrow)$ est une *approximation inférieure* de P par la récompense si

$$ R_\downarrow = \{(\phi_{lb}, r) \mid (\phi, r) \in R, r \geq 0\} \cup \{(\phi_{ub}, r) \mid (\phi, r) \in R, r < 0\} $$

où $\phi_{lb} \models \phi \models \phi_{ub}$. Les *approximations supérieures* $P_\uparrow$ sont définies de manière duale.

Théorème 2 *Soit P un PDM factorisé, $P_\downarrow$ et $P_\uparrow$ des approximations inférieures et supérieures de P. Alors, leurs fonctions de valeur optimales vérifient ${V_\downarrow}^n(s) \leq V^n(s) \leq {V_\uparrow}^n(s)$ pour tout état $s \in \mathcal{S}$ et horizon n.*

Démonstration. Par symétrie, nous montrons seulement ${V_\downarrow}^n(s) \leq V^n(s)$ car P peut être considéré comme une approximation inférieure de $P_\uparrow$. Nous procédons par récurrence sur l'horizon n. On a par définition :

$$\begin{aligned} {V_\downarrow}^0(s) = R_\downarrow(s) = & \sum \{r \mid (\phi_{lb}, r) \in R_\downarrow, s \models \phi_{lb}, r \geq 0\} \\ + & \sum \{r \mid (\phi_{ub}, r) \in R_\downarrow, s \models \phi_{ub}, r < 0\} \end{aligned}$$

Or, par définition des bornes, si $s \models \phi_{lb}$, alors $s \models \phi$ et si $s \models \phi$, alors $s \models \phi_{ub}$. Donc

$$\begin{aligned} \{r \mid (\phi_{lb}, r) \in R_\downarrow, s \models \phi_{lb}, r \geq 0\} &\subseteq \{r \mid (\phi, r) \in R, s \models \phi, r \geq 0\} \\ \{r \mid (\phi_{ub}, r) \in R_\downarrow, s \models \phi_{ub}, r < 0\} &\supseteq \{r \mid (\phi, r) \in R, s \models \phi, r < 0\} \end{aligned}$$

On en déduit finalement ${V_\downarrow}^0(s) \leq V^0(s)$.

À horizon $n+1$: en utilisant l'hypothèse de récurrence ${V_\downarrow}^n(s) \leq V^n(s)$ et $R_\downarrow(s) \leq R(s)$ (montré ci-dessus). On déduit, pour toute action a

$$R_\downarrow(s) + \gamma \sum_{e^a_{i,j}} p^a_{i,j} \cdot {V_\downarrow}^n(\mathit{ApplS}(e^a_{i,j}, s)) \;\leq\; R(s) + \gamma \sum_{e^a_{i,j}} p^a_{i,j} \cdot V^n(\mathit{ApplS}(e^a_{i,j}, s)).$$

C'est-à-dire ${Q_\downarrow}_a^{n+1}(s) \leq Q_a^{n+1}(s)$. D'où, pour tout état s, ${V_\downarrow}^{n+1}(s) \leq V^{n+1}(s)$. □

5.3 Encadrement des conditions d'actions

Pour un PDM factorisé P muni d'un ensemble d'actions $\mathcal{A}$, les problèmes approchés $P_\downarrow$ et $P_\uparrow$ sont obtenus à partir de P en remplaçant l'ensemble d'actions $\mathcal{A}$ par des ensembles $\mathcal{A}_\downarrow = \{a_\downarrow \mid a \in \mathcal{A}\}$ et $\mathcal{A}_\uparrow = \{a_\uparrow \mid a \in \mathcal{A}\}$, avec $a_\downarrow = (\{c^a_{i\ lb} \mid c^a_i \in \mathcal{C}^a\}, \mathcal{E}^a)$ et $a_\uparrow = (\{c^a_{i\ ub} \mid c^a_i \in \mathcal{C}^a\}, \mathcal{E}^a)$.

Nous nous restreignons dans cette section à des problèmes dont les récompenses sont positives. Intuitivement, il s'agit donc de rendre les actions exécutables dans plus ou moins d'états : si pour un état donné il y a moins d'actions possibles, alors sa valeur ne peut être plus grande, et inversement si plus d'actions sont possibles pour cet état. Dans la suite de cette section, nous allons discuter le cas des approximations inférieures et supérieures séparément.

Dans le cas des approximations inférieures, certains états peuvent n'avoir aucune action possible. Dans ce cas, en conformité avec l'équation 1, nous définissons la valeur espérée en cet état comme sa récompense immédiate, quel que soit l'horizon. La preuve suivante tient bien compte de ce cas.

Théorème 3 *Pour un PDM factorisé P dont toutes les récompenses sont positives ou nulles et son approximation inférieure sur les conditions d'actions $P_\downarrow$, la relation ${V_\downarrow}^n(s) \leq V^n(s)$ est vérifiée pour tout état s.*

Démonstration. Après initialisation par la récompense nous avons, pour tout état s, ${V_\downarrow}^0(s) = V^0(s) = R(s)$. Pour toute action a et son approximation $a_\downarrow$ nous avons

$$\{s \in \mathcal{S} \mid \mathit{Eff}(a_\downarrow, s) \neq \emptyset\} \subseteq \{s \in \mathcal{S} \mid \mathit{Eff}(a, s) \neq \emptyset\}$$

C'est-à-dire que $a_\downarrow$ est applicable sur moins d'états que a. L'espérance de récompense pour s en exécutant $a_\downarrow$ est identique à celle en exécutant a, ou nulle (donc inférieure à celle pour a puisque les récompenses sont positives).

Prenons maintenant pour hypothèse de récurrence $V_\downarrow^n(s) \leq V^n(s)$ alors :

$$R(s)+\gamma \sum_{Eff(a_\downarrow,s)} p_{i,j}^a {V_\downarrow}^n(ApplS(e_{i,j}^a, s)) \;\leq\; R(s)+\gamma \sum_{Eff(a,s)} p_{i,j}^a V^n(ApplS(e_{i,j}^a, s))$$

Et donc, pour tout état s, $V_\downarrow^{n+1}(s) \leq V^{n+1}(s)$. □

Notons que pour un problème $P = (\mathcal{S}, \mathcal{A}, T, R)$ et un autre $P_C = (\mathcal{S}, \mathcal{A}, T, R')$ tel que $R'(s) = R(s) + C$, C étant une constante, ainsi que leurs fonctions de valeur respectives V^n et V_C^n à horizon n nous avons l'égalité suivante :

$$V_C^n(s) = C\sum_{i=0}^{n-1}\gamma^i + V^n(s) = C\frac{1-\gamma^n}{1-\gamma} + V^n(s)$$

S'il existe une récompense négative, il est donc possible d'ajouter une constante C telle que :

$$C = -\min\{r \mid (\phi, r) \in R\}$$

à toutes les récompenses afin qu'elles soient positives et donc que le problème rentre dans le cadre d'application du théorème 3. Dans ce cas la politique reste inchangée, mais il faut soustraire $C\frac{1-\gamma^n}{1-\gamma}$ à la fonction de valeur obtenue pour retrouver celle du problème avec les récompenses originales.

Pour la borne supérieure $P_\uparrow$ sur les conditions d'actions d'un problème P, il se peut que $P_\uparrow$ ne soit pas un PDM. En effet, en relaxant les conditions il se peut que celles-ci ne soient plus mutuellement exclusives. Dans une telle situation, pour une action a et un état s, plusieurs condititions de a peuvent satisfaire s, ce qui aurait pour conséquence que les probabilités de transitions à partir de s auraient une somme supérieure à 1.

Ceci n'est pas nécessairement un obstacle, car ce problème peut être contourné en décomposant les actions : on crée de nouvelles actions en séparant les conditions dont les bornes ne sont pas mutuellement inconsistantes. Ce problème est alors résolu normalement, puis à la génération de la politique, les actions précédemment introduites sont regroupées pour revenir à l'espace d'actions originel.

Avec une preuve similaire à celle du théorème 3 nous avons le résultat suivant :

Théorème 4 *Pour un PDM factorisé P et son approximation supérieure sur les conditions d'actions $P_\uparrow$, telles que les conditions de chaque action de $P_\uparrow$ soient mutuellement exclusives*[3]*, la relation $V^n(s) \leq V_\uparrow^n(s)$ est vérifiée pour tout état s.*

3. Dans ce théorème il n'y a pas de restriction sur le signe des récompenses.

5.4 Approximation des effets

Il semble naturel qu'après l'approximation de la fonction de récompense et des conditions d'actions, nous nous intéressions au cas des effets. Cependant, de manière générale, il n'est pas possible de déduire un encadrement de la fonction de valeur à partir d'une approximation des effets, comme le montre l'exemple 7.

Exemple 7 *Soit un effet* $e = x \wedge \neg y$ *et sa borne supérieure* $e_\uparrow = x$*, et une fonction de valeur initiale* V^0 *telle que* $V^0(xyz) = 1$; $V^0(x\overline{y}\overline{z}) = 2$; $V^0(x\overline{y}z) = 3$; $V^0(xy\overline{z}) = 4$.

Pour les deux états $\overline{x}yz$ *et* $xy\overline{z}$ *nous obtenons les inégalités de valeur suivantes :*

$$\begin{aligned} V^0(ApplS(e, \overline{x}yz)) = 3 &> V^0(ApplS(e_\uparrow, \overline{x}yz)) = 1 \\ V^0(ApplS(e, xy\overline{z})) = 2 &< V^0(ApplS(e_\uparrow, xy\overline{z})) = 4 \end{aligned}$$

6 Application à la 2-CNF

La classe des formules 2-CNF (des conjonctions de clauses contenant au plus 2 littéraux) est intéressante pour la résolution factorisé de PDM. En effet, pour cette classe le test de satisfaisabilité est linéaire en la taille de la formule [1]. Pour l'oubli de variables il suffit de faire la conjonction de tous les impliqués premiers de la formule ne mentionnant pas de variable à oublier [8] ; leur nombre étant quadratique pour une formule en 2-CNF, il en résulte que l'oubli est en temps polynomial. Les formules en 2-CNF étant stables pour la conjonction, le théorème 1 garantit que les formules seront maintenues en 2-CNF tout au long de la résolution.

Cette classe n'est pas stable pour la négation, ni la disjonction. Il en résulte que l'initialisation des Q-valeurs de chaque action a dans l'algorithme 4 avec une formule $\phi = \neg(c^a_1 \vee \cdots \vee c^a_{n_a})$ peut poser problème. Il est possible de le contourner en remarquant que ϕ peut être précalculée. Cependant le langage des 2-CNF étant incomplet, il n'existe pas nécessairement de 2-CNF pouvant représenter ϕ. Malgré tout, le langage composé de disjonctions de 2-CNF est complet [6], il est alors possible de calculer l'ensemble des 2-CNF équivalent à ϕ et d'ajouter chacune d'elles dans la Q-valeur avec une valeur de $-\infty$. Pour cela, il faut recouvrir $\mathcal{M}(\phi)$ par un ensemble $\{M_1, \ldots, M_k\}$ tel que chaque M_i soit l'ensemble des modèles d'une formule 2-CNF, ce qui peut être déterminé en testant si M_i est clos par *majorité* [9]. Ensuite sur chaque M_i est appliqué un algorithme de *description* [11] qui va calculer une formule 2-CNF ayant pour modèles M_i.

Les définitions suivantes caractérisent les bornes minimales et maximales : celles qui encadrent « au plus près » la formule initiale. C'est en effet ce type de bornes qui est le plus intéressant car elles permettent d'approcher au mieux le problème original.

Définition 3 (Borne inférieure 2-CNF maximale : GLB) *Soit* ϕ *une formule quelconque. La formule 2-CNF* ϕ_{glb} *est une borne inférieure maximale de*

ϕ si et seulement si $\mathcal{M}(\phi_{glb}) \subseteq \mathcal{M}(\phi)$ et il n'existe pas de formule 2-CNF ϕ' telle que $\mathcal{M}(\phi_{glb}) \subset \mathcal{M}(\phi') \subseteq \mathcal{M}(\phi)$.

Définition 4 (Borne supérieure 2-CNF minimale : LUB) *Soit ϕ une formule quelconque. La formule 2-CNF ϕ_{lub} est une borne supérieure minimale de ϕ si et seulement si $\mathcal{M}(\phi) \subseteq \mathcal{M}(\phi_{lub})$ et qu'il n'existe pas de formule 2-CNF ϕ' telle que $\mathcal{M}(\phi) \subseteq \mathcal{M}(\phi') \subset \mathcal{M}(\phi_{lub})$. Pour une formule ϕ donnée, il existe une unique borne supérieure minimale, possiblement équivalente à ϕ [5].*

Exemple 8 *Soit la formule $\phi = (x \vee y \vee \neg z) \wedge (z \vee w)$. La formule $(x \vee y) \wedge (z \vee w)$ est une borne inférieure 2-CNF maximale de ϕ et $z \vee w$ en est une borne supérieure 2-CNF minimale. Ces deux formules vérifient bien $(x \vee y) \wedge (z \vee w) \models (x \vee y \vee \neg z) \wedge (z \vee w) \models z \vee w$*

Il existe des algorithmes, décrits dans [10], pour calculer ces bornes à partir de formules représentées en CNF, mais leur importante complexité ne permet d'envisager l'approximation de formules du problème de décision que comme une étape de compilation *a priori*. La description des actions représente la dynamique de l'agent et de l'environnement dans lequel il évolue. Sa mission est représentée quant à elle par la fonction de récompense. Le coût de l'approximation des actions peut donc être amorti dans le cas où plusieurs missions sont à effectuer. Chaque nouvelle mission ne requiert alors que l'approximation de la fonction de valeur initiale associée à la récompense décrivant la mission. L'approximation des formules de récompense doit alors se faire dans la classe des termes de longueur 2 pour que V^0 soit en 2-CNF.

7 Utilisation « anytime » des bornes inférieures

On peut imaginer compenser la perte de qualité des fonctions de valeur obtenue par approximation inférieure en utilisant plusieurs approximations successivement, en choisissant pour chaque ensemble d'états la plus grande valeur donnée par ces approximations. Les théorèmes 3 et 2 garantissent que cette approche est correcte.

Intuitivement, combiner ainsi des approximations revient à couvrir au mieux les conditions d'action, tout en maintenant des « petites » formules, afin de prendre en compte le plus de spécificités possibles du problème. La propriété *anytime* s'obtient donc en utilisant de plus en plus de bornes inférieures sur la fonction de valeur, en fonction du temps disponible.

Le choix des approximations à utiliser revêt une importance toute particulière. Tout d'abord, pour des raisons d'efficacité de génération des approximations, nous proposons d'utiliser des bornes inférieures non nécessairement maximales. D'autre part, afin de ne pas générer des approximations trop similaires du problème, nous choisissons des approximations distantes les unes des autres.

Dans le cas de la 2-CNF, l'approximation inférieure d'une CNF quelconque s'obtient par *renforcement* des clauses : on retire tout simplement des littéraux

de celles-ci pour que leur taille atteigne 2. Avec un tel procédé, on peut générer les approximations dans un ordre lexicographique, choisir les plus distantes selon cet ordre semble être un choix raisonnable pour représenter aux mieux les différents modèles de la formule originale car les formules obtenues auront peu de clauses en commun.

8 Résultats expérimentaux

Ne pouvant donner une borne théorique sur l'impact de ces approximations sur la fonction de valeur, nous avons mesuré l'écart de valeur sur des problèmes aléatoires. Nous discutons en premier lieu de la génération aléatoire de problèmes structurés dans notre représentation. Puis nous comparons les fonctions de valeur obtenues avec l'approximation à celles optimales.

8.1 Génération aléatoire de PDM structurés

Nous avons conçu un générateur de problèmes structurés aléatoires. Dans un premier temps la taille de l'espace d'états est définie par le nombre n de variables propositionelles utilisées. Ensuite, des actions sont générées : une formule CNF satisfaisable est générée aléatoirement sur un ensemble de k variables tiré aléatoirement, les clauses sont de taille comprise entre 2 et $\frac{k}{2}$, pour un nombre de clauses égal à k afin d'avoir une quantité relativement importante d'états représentés par cette formule. Cette clause va servir de base aux conditions : elle représente tous les états dans lesquels l'action est exécutable. En fonction du nombre de conditions voulu, t autres variables sont choisies aléatoirement pour former 2^t termes qui, chacun en conjonction avec la clause de base, permettront de partitionner celle-ci.

Pour chacune de ces conditions, un ensemble de e effets munis chacun d'une probabilité est généré. Pour cela, $e-1$ variables sont tirées aléatoirement, dont la moitié proviennent des conditions d'actions générées précédemment. Ce choix de variables permet d'activer les conditions d'autres actions afin de diversifier les politiques et de complexifier le problème. Un premier effet (terme) de taille $e-1$ est généré, ainsi que sa probabilité ; pour les effets suivants, un littéral est retiré de l'effet précédent jusqu'à obtenir l'effet vide. Cette génération se base sur l'hypothèse que les différentes issues d'une action sont les mêmes à la différence près qu'elle peut dans certains cas ne pas fonctionner totalement, ou même n'avoir aucun effet du tout.

Les récompenses sont tout simplement des littéraux, munies d'une récompense aléatoire positive. Le nombre de formules de récompenses est paramétrable et permet de définir combien il y aura de récompenses différentes possibles.

Exemple 9 (génération aléatoire d'une action) *Prenons pour paramètres $n = 10$ variables, $e = 3$ effets, $c = 2$ conditions sur $k = 6$ variables par action.*

1. *Génération d'une CNF aléatoire sur k variables :*

$$\phi = (x_0 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee \neg x_5)$$

2. *On choisit* $\log c$ *variables pour partitionner* ϕ *et obtenir les conditions : ici 1 variable, par exemple* x_6*. On obtient deux conditions mutuellement inconsistantes :*
   - $C_1 = (x_0 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge x_6$
   - $C_2 = (x_0 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge \neg x_6$
3. *Pour chacune des conditions on génère les effets et leurs probabilités :*
   - *pour* C_1 *:* $(\{x_2, \neg x_4\},\ 0,6), (\{x_2\},\ 0,3), (\emptyset,\ 0,1)$,
   - *pour* C_2 *:* $(\{\neg x_3, x_6\},\ 0,5), (\{x_6\},\ 0,2), (\emptyset,\ 0,1)$.

8.2 Résultats

Des PDM ont été générés de la façon présentée précédemment, avec leurs approximations pour obtenir un problème en 2-CNF. Pour chaque problème nous avons calculé la fonction de valeur et une politique optimales, ainsi que les politiques pour les problèmes approchés au même horizon que pour le problème initial. Ces deux politiques approchées ont ensuite été évaluées sur le problème original afin de déterminer leur fonction de valeur associée et de la comparer à l'optimale.

Nous avons réalisé un solveur de PDM factorisés faisant intervenir la bibliothèque CUDD [4] pour représenter les formules propositionnelles quelconques sous forme de diagrammes de décision binaires (BDD) ainsi qu'une implémentation des formules 2-CNF avec leurs opérations associées.

Les temps présentés pour la résolution des problèmes approchés sont obtenus en utilisant une représentation en BDD. En effet, il se trouve que la bibliothèque CUDD est plus efficace en pratique qu'une implémentation dédiée des classes de formules malgré la complexité théorique polynomiale des opérations. De plus, la description des états où une action n'est pas applicable n'est plus nécessaire puisque les BDD peuvent représenter des formules arbitraires.

8.2.1 Approximation en 2-CNF

Pour la résolution sur les problèmes aux conditions d'action approchées par des bornes maximales en 2-CNF nous avons résolu 10 problèmes aléatoires comportant chacun 16 variables (soit 65536 états) et 10 actions. Pour chaque problème présenté dans le tableau 2, les trois premières lignes donnent la taille de la fonction de valeur, le temps de résolution (calcul de l'approximation compris) et la qualité moyenne de la politique obtenue pour les problèmes exacts, approchés par une LUB en 2-CNF et approchés par une GLB en 2-CNF, respectivement.

Problème	Approx. (Temps)	$\lvert V \rvert$	T	(%)	RM	(%)
	-	48527	333	(100)	250,92	(100)
0						

4. `http://vlsi.colorado.edu/~fabio/CUDD/`

Problème	Approx. (Temps)	$\|V\|$	T	(%)	RM	(%)
	LUB 2-CNF (<1)	40968	306	(91,89)	87,86	(35,02)
	GLB 2-CNF (1)	43495	232	(69,67)	201,15	(80,16)
	GLB Horn (<1)	41131	221	(66,37)	207,01	(82,5)
1	-	31614	199	(100)	305,89	(100)
	LUB 2-CNF (<1)	20055	139	(69,85)	152,09	(49,72)
	GLB 2-CNF (1)	24999	124	(62,31)	248,3	(81,17)
	GLB Horn (<1)	30332	143	(71,86)	258,75	(84,59)
2	-	39735	258	(100)	288,35	(100)
	LUB 2-CNF (<1)	32711	268	(103,88)	122,35	(42,43)
	GLB 2-CNF (2)	35574	208	(80,62)	241,67	(83,81)
	GLB Horn (<1)	30969	146	(56,59)	232,76	(83,31)
3	-	49742	384	(100)	253,56	(100)
	LUB 2-CNF (<1)	47587	451	(117,45)	134,92	(53,21)
	GLB 2-CNF (2)	44273	276	(71,88)	215,94	(85,16)
	GLB Horn (<1)	43341	240	(62,5)	188,3	(80,19)
4	-	33581	222	(100)	288,34	(100)
	LUB 2-CNF (<1)	28673	201	(90,54)	166,31	(56)
	GLB 2-CNF (1)	32968	169	(76,13)	254,62	(85,74)
	GLB Horn (<1)	25466	130	(58,56)	238,13	(86,47)
5	-	26190	150	(100)	239,64	(100)
	LUB 2-CNF (<1)	21715	143	(95,33)	122,21	(51)
	GLB 2-CNF (1)	23133	117	(78)	199,43	(83,22)
	GLB Horn (<1)	21929	100	(66,67)	207,22	(79,04)
6	-	17749	86	(100)	264,36	(100)
	LUB 2-CNF (<1)	10753	63	(73,26)	134,66	(50,94)
	GLB 2-CNF (1)	11613	73	(84,88)	221,02	(83,61)
	GLB Horn (<1)	12877	44	(51,16)	208,96	(79,04)
7	-	34559	224	(100)	260,44	(100)
	LUB 2-CNF (<1)	21761	148	(66,07)	148,45	(57)
	GLB 2-CNF (2)	42307	168	(75)	220,02	(85,55)
	GLB Horn (<1)	31814	171	(76,34)	232,04	(89,1)
8	-	25729	168	(100)	262,82	(100)
	LUB 2-CNF (<1)	13800	95	(56,55)	116,22	(44,22)

Problème	Approx. (Temps)	$\|V\|$	T	(%)	RM	(%)
	GLB 2-CNF (2)	26288	128	(76,19)	207,81	(79,07)
	GLB Horn (<1)	30116	71	(86,9)	211,58	(80,5)
9	-	20862	109	(100)	274,35	(100)
	LUB 2-CNF (<1)	18890	121	(111,01)	231,13	(84,25)
	GLB 2-CNF (2)	20353	87	(79,82)	228,26	(83,2)
	GLB Horn (<1)	18937	71	(65,14)	234,32	(85,41)
Moyennes	-	32828,8	213,3	(100)	269,73	(100)
	LUB 2-CNF (<1)	25691,3	193,5	(87,58)	141,62	(52,38)
	GLB 2-CNF	30500,3	158,2	(75,45)	223,84	(82,97)
	GLB Horn (<1)	28691,2	141,2	(66,21)	221,91	(82,28)

TABLE 2 – Temps de calcul et qualité d'approximation par bornes inférieures et supérieures maximales en 2-CNF, sur des problèmes à 16 variables et 10 actions, à horizon 10. Légende : Approx. (Temps) : Type d'approximation et temps de calcul ; $|V|$: taille de la partition de valeurs finale en nombre de formules ; T : temps de résolution, (avec approximation), en secondes ; RM : récompense moyenne d'un état

Sans surprise, les valeurs obtenues en évaluant les politiques obtenues grâce aux approximations supérieures minimales sont décevantes car certainement trop « optimistes ». De plus, le gain en temps de calcul est marginal.

Concernant les approximations inférieures maximales, les résultats sont plus réguliers : on observe sur tous les problèmes un gain en temps de calcul de l'ordre de 25 % pour une perte de qualité de l'ordre de 20 %. Si la perte de qualité reste acceptable, le gain en temps de calcul n'est pas significatif. Des expériences sur des problèmes plus grands ont montré une perte de qualité similaire pour un temps de résolution plus mauvais.

Notons que pour ces problèmes de taille raisonnable, le temps de calcul des approximations est négligeable vis-à-vis du temps de résolution.

8.2.2 Approximation en formules de Horn

Au vu des résultats décevants pour la 2-CNF, qui ne bénéficie pas expérimentalement de son efficacité théorique nous avons mené des expériences avec la classe des formules de Horn (CNF avec au plus 1 littéral positif par clause). La notion d'encadrement de la fonction de valeur présentée an section 5 étant valide pour toute classe de formules, cette technique reste en effet correcte.

Nous avons conservé l'utilisation de BDD dans notre solveur, plutôt qu'une implémentation dédiée des opérations logiques sur les formules de Horn. Une telle implémentation s'est en effet révélée peu efficace pour la 2-CNF alors même que la complexité de l'oubli est meilleure que celle de Horn. En outre, au vu des résultats de la 2-CNF nous n'avons pas testé les bornes supérieures.

Problème	Approx.	$\|V\|$	T	(%)	RM	(%)
0	-	72552	261	(100)	175,51	(100)
	GLB Horn	80928	226	(86,59)	160,83	(91,64)
1	-	197903	966	(100)	149,65	(100)
	GLB Horn	119649	357	(36,96)	120,78	(80,71)
2	-	78853	287	(100)	121,22	(100)
	GLB Horn	41593	83	(28,92)	100,58	(82,97)
3	-	91110	275	(100)	136,83	(100)
	GLB Horn	82316	168	(61,09)	118,78	(86,81)
4	-	69471	316	(100)	133,14	(100)
	GLB Horn	70253	258	(81,65)	122,25	(91,82)
5	-	141634	780	(100)	113,5	(100)
	GLB Horn	102656	359	(46,03)	104,82	(92,35)
6	-	75223	230	(100)	138,64	(100)
	GLB Horn	75563	152	(66,09)	113,84	(82,11)
7	-	205181	1133	(100)	260,44	(100)
	GLB Horn	101377	359	(31,69)	232,04	(89,1)
8	-	76542	259	(100)	262,82	(100)
	GLB Horn	70313	189	(72,97)	211,58	(80,5)
9	-	65497	194	(100)	274,35	(100)
	GLB Horn	49541	117	(60,31)	234,32	(85,41)
Moyennes	-	107396,6	470,1	(100)	176,61	(100)
	GLB Horn	79418,9	226,8	(57,23)	151,98	(86,34)

TABLE 3 – Temps de calcul et qualité d'approximation par bornes inférieures maximales en 2-CNF et Horn, sur des problèmes aléatoires à 20 variables et 12 actions à horizon 6

Le tableau 2 présente les résultats sur les mêmes problèmes que pour la 2-CNF, et les tableaux 3 et 4 comparent les temps de calcul avec le problème exact et approché par une borne inférieure maximale de Horn pour des problèmes plus importants, respectivement 20 variables, 12 actions, et 24 variables, 15 actions.

On constate sur ces approximations une perte de qualité de l'ordre de 15 à 20 % mais cette fois le gain de temps de calcul est significatif : entre 35 et 45 %. Il semblerait que cette approche soit d'autant plus efficace que le problème est grand, tendance qui reste à confirmer par d'autres expériences. Notons que les résultats montrent une certaine robustesse, car ces gains se retrouvent sur toutes les instances testées.

8.2.3 Approximations successives

Nous avons également testé l'approche par approximations successives du section 7, avec des 2-CNF. La figure 2 montre l'évolution de la récompense moyenne des politiques générées en fonction du temps, chaque palier représen-

Problème	Approx.	$\|V\|$	T	(%)	RM	(%)
0	-	422089	2650	(100)	88,29	(100)
	GLB Horn	268890	1004	(37,89)	73,9	(83,7)
1	-	162033	706	(100)	133,24	(100)
	GLB Horn	137572	416	(58,92)	113,47	(85,16)
2	-	214643	710	(100)	96,32	(100)
	GLB Horn	186229	439	(61,83)	85,56	(88,83)
3	-	362938	1563	(100)	83,14	(100)
	GLB Horn	170185	871	(55,73)	66,79	(80,33)
4	-	227090	1166	(100)	111,94	(100)
	GLB Horn	186229	710	(60,89)	95,97	(85,73)
Moyennes	-	277758,6	1359	(100)	102,59	(100)
	GLB Horn	189821	688	(55,05)	87,14	(84,75)

TABLE 4 – Temps de calcul et qualité d'approximation par bornes inférieures maximales de Horn, sur des problèmes aléatoires à 24 variables et 15 actions à horizon 4

tant une nouvelle approximation. Le problème testé comporte 20 variables et 12 actions. On constate que la qualité augmente bien avec le temps, mais que pour un temps proche de celui de la résolution exacte la perte de qualité est trop importante. Ceci a été confirmé sur d'autres instances avec la 2-CNF et Horn.

L'explication semble être la suivante. Pour que le temps de calcul des approximations soit raisonnable, il faut utiliser des bornes inférieures non nécessairement maximales. Ceci résulte en une perte de qualité trop importante, comme le montre le premier palier de la figure 2 : 50 % de qualité tandis que les bornes maximales montrent une qualité de l'ordre de 80 % (tableau 2).

9 Conclusion et travaux futurs

Dans ces travaux nous avons proposé une méthode de résolution de processus de décision markoviens représentés de manière factorisée à l'aide de la logique propositionnelle. À partir de cette représentation nous avons étudié l'impact des classes de formules dites efficaces sur la complexité de la résolution. Par la suite nous avons proposé une méthode d'approximation des problèmes de décision basée sur l'encadrement de formules propositionnelles, une méthode issue de travaux sur l'approximation et la compilation de bases de connaissances.

Notre étude de la 2-CNF a montré que l'efficacité théorique de ses opérations ne s'est pas reflétée en pratique, car l'utilisation d'un solveur manipulant des formules sous la forme de BDD s'est révélée plus efficace qu'une implémentation spécifique pour cette classe de formules. De plus, le gain expérimental n'est pas significatif sur les problèmes aléatoires testés. De la même manière l'approche

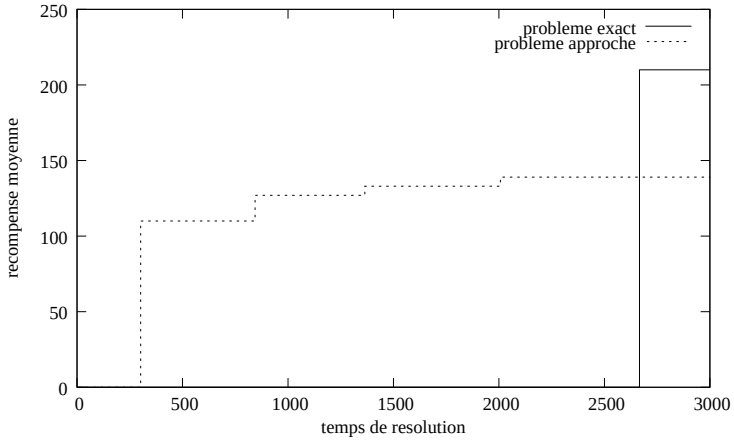

FIGURE 2 – Qualité des approximations sucessives et temps de résolution par bornes inférieures en 2-CNF

« anytime », pourtant séduisante en théorie, s'avère peu intéressante en pratique. Néanmoins elle reste à expérimenter de manière parallèlle en calculant différents minorants ou majorants de manière simultanée.

Ces résultats décevants sont compensés par des résultats prometteurs avec des approximations dans la classe des formules de Horn. En effet, notre méthode permet un gain de temps de résolution de presque moitié pour une perte de qualité inférieure à 20 % sur les problèmes testés.

Bien que la qualité des approximations est en deçà de nos espérances, celles-ci pourraient s'avérer utiles en tant qu'heuristiques pour des algorithmes tels que SLAO* et sRTDP [**?**, **?**]. Ceux-ci requièrent en effet une fonction de valeur initiale qui est un majorant de la fonction de valeur optimale.

Le gain en temps de calcul semble venir de la plus grande simplicité des formules et donc des BDD manipulés. Cependant nous n'avons pas de résultat théorique appuyant cette intuition ; par exemple, dans le cas des 2-CNF il existe des BDD de taille exponentielle bien qu'équivalents à une 2-CNF. Quoi qu'il en soit, les résultats expérimentaux restent similaires sur les diverses instances testées, ce qui montre une certaine robustesse de notre approche.

D'un point de vue théorique, notre approche d'approximation de problèmes fournit un cadre solide pour l'étude d'autres biais de représentation des PDM factorisés. Citons par exemple l'approximation sur un sous-ensemble de variables. De manière différente à d'autres approches de simplification des problèmes par élimination de variables, telles que [4], notre cadre permet seulement d'obtenir un encadrement de la fonction de valeur réelle. Des expériences restent à mener en ce sens.

Une autre perspective consiste à tester notre méthode sur des problèmes réels. Typiquement, de tels problèmes exhibent plus de structure que les pro-

blèmes aléatoires, et pourraient de fait bénéficier d'autant plus des approximations dans des classes de formules qui capturent ces structures.

Enfin, de manière générale, notre approche de résolution de PDM factorisés permet de traiter naturellement des actions dont les effets portent sur plusieurs variables. Ceci est une différence essentielle avec l'algorithme SPUDD [7], qui ne traite de tels problèmes qu'au prix d'une représentation moins naturelle (*via* les « arcs synchrones »). Néanmoins, le solveur SPUDD est plus efficace que le nôtre en pratique. Nous comptons donc y intégrer notre méthode d'approximation, et déterminer par l'expérience s'il peut bénéficier également de notre technique.

Références

[1] B. Aspvall, M. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3) :121–123, 1979.

[2] R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[3] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, 1995.

[4] R. Dearden and C. Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1-2) :219–283, 1997.

[5] A. Del Val. An Analysis of Approximate Knowledge Compilation. In *International Joint Conference on Artificial IntelligenceI*, volume 14, pages 830–836, 1995.

[6] H. Fargier and P. Marquis. Extending the Knowledge Compilation Map : Krom, Horn, Affine and Beyond. *Proceedings of AAAI'08*, 2008.

[7] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD : Stochastic planning using decision diagrams. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288, 1999.

[8] J. Lang, P. Liberatore, and P. Marquis. Propositional Independence : Formula-Variable Independence and Forgetting. *Journal of Artificial Intelligence Research*, 18 :391–443, 2003.

[9] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth ACM symposium on Theory of computing*, pages 216–226. ACM Press, NY, USA, 1978.

[10] B. Selman and H. Kautz. Knowledge Compilation and Theory Approximation. *Journal of the Association for Computing Machinery*, 43(2) :193–224, 1996.

[11] B. Zanuttini and J.J. Hébrard. A unified framework for structure identification. *Information Processing Letters*, 81(6) :335–339, 2002.