Chapitre 1

Représentations factorisées

1.1. Introduction

L'ensemble des solutions décrites dans le cadre des MDP (chapitre **??**, volume 1), que ce soit pour la planification ou l'apprentissage par renforcement, partagent toutes un inconvénient commun : elles ne sont pas adaptées à la résolution de problèmes de grande taille. En effet, l'utilisation de représentations non structurées nécessite une énumération explicite de l'ensemble des états possibles du problème pour représenter les fonctions nécessaires à sa résolution.

EXEMPLE.– Dans le cas de la voiture qu'il faut entretenir (voir volume 1, section **??**), on conçoit rapidement que l'ensemble des états possibles d'une voiture peut devenir gigantesque. Par exemple, on peut tenir compte de l'état d'usure de chacune des pièces de la voiture. L'idée sous-tendant ce chapitre est que la voiture est constituée d'une collection de sous-systèmes plus ou moins indépendants. Par exemple, faire une vidange ne devrait pas, en principe, influencer l'état des freins de la voiture. Il semble alors possible de prendre en compte la relative indépendance de ces sous-systèmes pour décrire plus efficacement le modèle du problème dans l'espoir que la recherche d'une solution sera plus simple. Peut-être apprendrons-nous ainsi qu'il est toujours optimal de remplacer des freins usés, quel que soit l'état du reste de la voiture.

Ce chapitre vise donc à décrire une extension des MDP présentée par [BOU 95, BOU 99], appelée *processus décisionnels de Markov factorisés* (*Factored Markov Decision Processes* (FMDP)) et permettant de représenter les fonctions de transition et de récompense d'un problème de façon compacte (section 1.2). Une fois le problème

---

Chapitre rédigé par Thomas DEGRIS et Olivier SIGAUD.

représenté de façon compacte, nous décrirons plusieurs méthodes de planification permettant de trouver les solutions optimales ou optimales approchées (section 1.3), tout en exploitant la structure du problème afin d'éviter une énumération explicite de l'espace d'état. Nous conclurons section 1.4.

1.2. Le formalisme des FMDP

1.2.1. *Représentation de l'espace d'état*

Il est souvent naturel de décrire un problème par un ensemble de paramètres pouvant prendre différentes valeurs décrivant l'état courant du système. Ainsi, l'ensemble des états possibles peut être caractérisé par un ensemble de variables aléatoires. En pratique, l'ensemble des états possibles S est décrit par un ensemble de variables aléatoires $X = \{X_1, \ldots, X_n\}$ où chaque variable X_i peut prendre différentes valeurs dans son domaine $\text{DOM}(X_i)$. Un état est donc une instanciation de X décrite sous la forme d'un vecteur $x = \{x_1, \ldots, x_n\}$ de valeurs x_i avec $\forall i\ x_i \in \text{DOM}(X_i)$. De plus, on utilise comme raccourci d'écriture $\text{DOM}(X)$ pour décrire l'ensemble des instanciations possibles des variables $X_i \in X$. L'espace d'état S du MDP est donc $S = \text{DOM}(X)$.

L'avantage d'une telle représentation est qu'il est possible d'exploiter différentes structures existantes dans un problème pour, d'une part, le représenter de façon compacte, d'autre part le résoudre en limitant la complexité de la solution et de la méthode utilisée pour l'obtenir. Etant donnée une telle décomposition, les principales contributions du cadre mathématique des FMDP sont de *décomposer* les fonctions de transition et de récompense (respectivement de façon multiplicative et additive) afin d'exploiter *les indépendances relatives aux fonctions* liées à la structure du problème. De plus, les FMDP offrent un cadre approprié à l'utilisation, de façon complémentaire mais pas obligatoire, de deux autres propriétés liées à la structure d'un problème : *les indépendances relatives aux contextes* et *l'approximation additive*. Dans la suite, nous parlerons d'« indépendances fonctionnelles » (respectivement « contextuelles ») pour désigner les indépendances relative aux fonctions (respectivement aux contextes).

Afin d'illustrer le cadre des FMDP, nous utiliserons l'exemple du *Coffee Robot* proposé par [BOU 00] bien connu dans la littérature des FMDP. Une fois l'exemple *Coffee Robot* décrit (paragraphe 1.2.2), le paragraphe 1.2.3 décrit les décompositions des fonctions de transition et de récompense ainsi que la formalisation des indépendances fonctionnelles. Le paragraphe 1.2.4 propose une formalisation du concept d'indépendance contextuelles. Enfin, l'approximation additive sera étudiée plus tard, dans le contexte de son utilisation lors du paragraphe 1.3.3.5.

1.2.2. *L'exemple* Coffee Robot

Un robot doit aller acheter un café pour sa propriétaire restant au bureau. Quand il pleut, comme le robot doit sortir pour aller chercher le café, il doit se munir d'un parapluie lorsqu'il est au bureau, sinon il sera mouillé. Pour décrire l'état du système, six variables aléatoires binaires[1] ($\text{DOM}(X_i) = \{0, 1\}$ correspondant respectivement à `Faux` et `Vrai`) sont utilisées :

$\mathcal{HOC}$: la propriétaire a-t-elle un café ? *(Has Owner Coffee ?)*

$\mathcal{HRC}$: le robot a-t-il un café ? *(Has Robot Coffee ?)*

$\mathcal{W}$: le robot est-il mouillé ? *(Wet ?)*

$\mathcal{R}$: est-ce qu'il pleut ? *(Raining ?)*

$\mathcal{U}$: le robot a-t-il un parapluie ? *(Umbrella ?)*

$\mathcal{O}$: le robot est-il au bureau ? *(Office ?)*

Par exemple, le vecteur $[\mathcal{HOC}{=}0, \mathcal{HRC}{=}1, \mathcal{W}{=}0, \mathcal{R}{=}1, \mathcal{U}{=}0, \mathcal{O}{=}1]$ représente un état de ce problème dans lequel la propriétaire n'a pas de café, le robot a un café, le robot n'est pas mouillé, il pleut, le robot n'a pas de parapluie et le robot est au bureau. Ce problème étant composé de 6 variables binaires, son espace d'états contient $2^6 = 64$ états possibles.

Dans le problème *Coffee Robot*, le robot dispose de quatre actions possibles :

$\mathcal{G}\text{o}$: se déplacer vers le lieu opposé ;

$\mathcal{B}\text{uyC}$: acheter un café, que le robot obtient s'il est au café *(Buy Coffee)* ;

$\mathcal{D}\text{elC}$: donner le café à sa propriétaire, qu'elle peut obtenir si le robot est au bureau et qu'il a un café *(Deliver Coffee)* ;

$\mathcal{G}\text{etU}$: prendre un parapluie, que le robot peut obtenir s'il est au bureau *(Get Umbrella)*.

1. Principalement pour des raisons de simplicité d'exposition, la plupart des exemples décrits dans ce chapitre utilisent des variables binaires. Cependant, rien ne limite l'utilisation des méthodes exposées à des problèmes contenant des variables énumérées.

L'effet de ces actions peut être bruité afin de représenter les cas stochastiques. Par exemple, lorsque le robot donne la tasse de café à sa propriétaire, la propriétaire obtiendra son café avec une certaine probabilité. L'action peut mal se passer, par exemple, lorsque le robot renverse le café. Ainsi, lorsque le robot exécute l'action $\mathcal{D}\text{elC}$ dans l'état $s = [\mathcal{HOC}=0,\mathcal{HRC}=1,\mathcal{W}=0,\mathcal{R}=1,\mathcal{U}=0,\mathcal{O}=1]$ (le robot a un café et est au bureau, la propriétaire n'a pas de café), la fonction de transition définit :

– $P([\mathcal{HOC}=1,\mathcal{HRC}=1,\mathcal{W}=0,\mathcal{R}=1,\mathcal{U}=0,\mathcal{O}=1]|s, \mathcal{D}\text{elC}) = 0.8$,

– $P([\mathcal{HOC}=0,\mathcal{HRC}=1,\mathcal{W}=0,\mathcal{R}=1,\mathcal{U}=0,\mathcal{O}=1]|s, \mathcal{D}\text{elC}) = 0.2$,

– 0.0 pour les autres probabilités.

Enfin, le robot reçoit une récompense de 0.9 lorsque la propriétaire a un café (0 lorsqu'elle n'a pas de café) ajoutée à 0.1 lorsqu'il est sec (et 0 lorsqu'il est mouillé). La récompense obtenue lorsque la propriétaire a un café est supérieure à la récompense obtenue par le robot lorsqu'il reste sec pour indiquer que le premier objectif est prioritaire sur le deuxième. Dans cet exemple, la fonction de récompense ne dépend pas de l'action réalisée par le robot.

1.2.3. *Décomposition et indépendances relatives aux fonctions*

Les indépendances relatives aux fonctions expriment le fait que certaines définitions du problème ne dépendent pas systématiquement de toutes les autres variables du problème ou de l'action réalisée par l'agent. Par exemple, dans le problème *Coffee Robot*, la valeur de la variable $\mathcal{R}$ au prochain pas de temps – indiquant s'il pleut ou non – ne dépend que de sa propre valeur au pas de temps courant. En effet, le fait qu'il va pleuvoir au prochain pas de temps est indépendant des variables telles que « le robot a-t-il un café ? » (variable $\mathcal{HRC}$) ou de l'action exécutée par l'agent. Le cadre des FMDP permet d'exploiter cette propriété principalement dans la description des fonctions de transition et de récompense du problème et dans l'utilisation de ces fonctions par les algorithmes de planification. Cette notion est formalisée par deux opérateurs, PARENTS et SCOPE, qui sont définis respectivement dans le cadre de la représentation de la fonction de transition (paragraphe 1.2.3.1) et de la fonction de récompense (paragraphe 1.2.3.4).

1.2.3.1. *Décomposition de la fonction de transition*

En supposant que l'ensemble des états possibles se décompose en un ensemble de variables aléatoires (décrit paragraphe 1.2.1), il est possible de décomposer une probabilité $P(s'|s)$ en un produit de probabilités, puis d'exploiter les indépendances entre ces variables aléatoires afin de rendre plus compacte la description de la fonction de transition.

Par exemple, admettons qu'un espace d'état soit décrit avec trois variables binaires X, Y et Z. Pour énumérer l'ensemble des combinaisons possibles $P(s'|s)$, il est nécessaire de décrire une table contenant $2^{2\cdot 3} = 64$ entrées. En décomposant la probabilité $P(s'|s)$ en un produit de probabilités, on obtient :

$$\begin{aligned} P(s'|s) &= P(x', y', z'|s) \\ &= P(x'|s)P(y'|s, x')P(z'|s, x', y') \end{aligned}$$

avec x représentant la valeur de la variable X au pas de temps t et, x' la valeur de la variable X au pas de temps $t + 1$. De plus, si les relations de dépendance entre les variables sont connues, alors $P(s'|s)$ peut s'écrire de façon plus compacte. Par exemple, si chacune des variables X, Y et Z ne dépend que de sa valeur dans l'état précédent sauf la variable Y qui dépend aussi de X dans l'état précédent, alors :

$$\begin{aligned} P(s'|s) &= P(x'|s)P(y'|s, X')P(z'|s, x', y') \\ &= P(x'|x)P(y'|y, x)P(z'|z) \end{aligned}$$

En agrégeant les états pour lesquels la fonction de transition est identique, seules $2^1 + 2^2 + 2^1 = 8$ entrées sont nécessaires et réparties en trois tables différentes, une pour chaque variable (correspondant respectivement à la description des distributions de probabilité $P(X'|X)$, $P(Y'|Y, X)$ et $P(Z'|Z)$).

Ainsi, les indépendances fonctionnelles liées à la structure du problème sont mises en évidence et permettent ainsi d'agréger certaines régularités dans la description de la fonction de transition. De plus, elles correspondent à une représentation intuitive consistant à décrire l'effet de chaque action sur la valeur de chacune des variables du problème. Cette représentation de la fonction de transition est formalisée en utilisant le cadre des réseaux bayésiens dynamiques [BOU 95].

1.2.3.2. *Les réseaux bayésiens dynamiques*

Les réseaux bayésiens [PEA 88] sont un formalisme permettant de représenter graphiquement des dépendances (ou indépendances) entre des variables. Les variables constituent les nœuds d'un graphe orienté, les relations directes de dépendance probabiliste entre deux variables sont représentées par un arc entre les deux nœuds représentant chacun des variables. Les réseaux bayésiens dynamiques [DEA 89] (*Dynamic Bayesian Networks (*DBN*)*) sont des réseaux bayésiens permettant de représenter les données temporelles engendrées par des processus stochastiques.

En faisant l'hypothèse que le problème observé est stationnaire (donc la fonction de transition T du MDP ne varie pas au cours du temps), il est possible de représenter T avec des DBN faisant seulement apparaître deux pas de temps successifs. Dans ce cas, les DBN sont composés de deux ensembles de nœuds :

1) l'ensemble de nœuds représentant l'ensemble des variables de l'espace d'état à l'instant t ;

2) l'ensemble de nœuds représentant l'ensemble des variables de l'espace d'état à l'instant $t+1$.

Les arcs indiquent alors les dépendances directes entre les variables à l'instant t et les variables à l'instant $t+1$ ou encore des dépendances entre les variables à l'instant $t+1$ (ces arcs sont appelés arcs synchrones). Dans ce cas particulier, un DBN est quelque fois appelé *2 Time Bayesian Network*. Pour une question de clarté, nous ferons l'hypothèse que les arcs synchrones ne sont pas nécessaires pour décrire le modèle des transitions du problème.

Il est alors possible de représenter graphiquement les dépendances de la fonction de transition en utilisant un DBN par variable et par action. L'action exécutée par l'agent peut aussi être considérée comme une variable à l'instant t. Dans ce cas, un seul DBN par variable suffit [BOU 96]. Enfin, pour être complet, un DBN doit être quantifié, comme illustré dans la section suivante pour l'exemple *Coffee Robot*.

1.2.3.3. *Modèle factorisé de la fonction de transition*

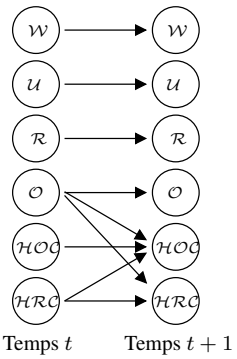
La figure 1.1 montre la représentation de l'effet de l'action $\mathcal{D}\mathrm{elC}$ sur l'ensemble des états. Le DBN (figure 1.1(a)) permet de constater facilement que, pour l'action $\mathcal{D}\mathrm{elC}$, la variable $\mathcal{HOC}$ ne dépend que des variables $\mathcal{O}$, $\mathcal{HRC}$ et $\mathcal{HOC}$ au pas de temps précédent et est indépendante des autres variables du problème. On définit $\textsc{Parents}_\tau(X'_i)$ l'ensemble des parents de la variable X'_i dans ce DBN τ. Cet ensemble peut être partitionné en deux sous-ensembles $\textsc{Parents}^t_\tau(X'_i)$ et $\textsc{Parents}^{t+1}_\tau(X'_i)$ représentant respectivement l'ensemble des parents au temps t et l'ensemble des parents au temps $t+1$. Nous supposons l'absence d'arc synchrone, donc $\textsc{Parents}^{t+1}_\tau(X'_i) = \emptyset$ et $\textsc{Parents}_\tau(X'_i) = \textsc{Parents}^t_\tau(X'_i)$. Dans l'exemple de la figure 1.1, nous avons $\textsc{Parents}_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}') = \{\mathcal{O}, \mathcal{HRC}, \mathcal{HOC}\}$.

Afin de quantifier l'effet d'une action sur l'espace d'états, on spécifie la probabilité $P_\tau(X'_i|x)$ pour chaque instanciation possible $x \in \textsc{Dom}(\textsc{Parents}_\tau(X'_i))$. Chaque réseau d'action DBN τ est donc quantifié par un ensemble de *distributions de probabilités conditionnelles*[2]. On note $P_\tau(X'_i|\textsc{Parents}_\tau(X'_i))$ une telle distribution pour une variable X'_i. Une probabilité $P_\tau(s'|s)$ de la fonction de transition peut alors être définie de façon compacte :

$$P_\tau(s'|s) = \prod_i P_\tau(x_i^{s'}|x^s) \tag{1.1}$$

avec $x_i^{s'}$ l'instanciation de la variable X'_i dans l'état s' et x^s l'instanciation des variables appartenant à $\textsc{Parents}_\tau(X'_i)$.

2. *Conditional Probability Distributions.*



(a)

$\mathcal{HOC}$	$\mathcal{HRC}$	$\mathcal{O}$	$\mathcal{HOC}'$
1	1	1	1.0
1	1	0	1.0
1	0	1	1.0
1	0	0	1.0
0	1	1	0.8
0	1	0	0.0
0	0	1	0.0
0	0	0	0.0

(b)

Figure 1.1. *Représentation (partielle) de la fonction de transition T pour le problème* Coffee Robot*. La figure (a) représente les dépendances entre les variables pour l'action $\mathcal{D}$elC sous la forme d'un* DBN*. La figure (b) définit la distribution de probabilité conditionnelle $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O},\mathcal{HRC},\mathcal{HOC})$ sous forme tabulaire.*

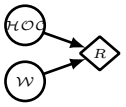
La figure 1.1(b) représente sous forme tabulaire et pour l'action $\mathcal{D}$elC la distribution de probabilité conditionnelle $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O},\mathcal{HRC},\mathcal{HOC})$ dans le problème *Coffee Robot*. Les colonnes $\mathcal{O}$, $\mathcal{HRC}$ et $\mathcal{HOC}$ représentent la valeur de ces variables à l'instant t, la colonne $\mathcal{HOC}'$ représente la probabilité pour la variable $\mathcal{HOC}$ d'avoir la valeur `Vrai` au temps $t+1$.

La décomposition multiplicative et l'exploitation des indépendances fonctionnelles dans la description du modèle des transitions et dans le calcul des probabilités qui en découlent sont les principales contributions des FMDP par rapport aux MDP. Ces deux propriétés sont exploitées par l'ensemble des algorithmes décrits dans le cadre des FMDP.

1.2.3.4. *Modèle factorisé de la fonction de récompense*

Pour spécifier complètement un MDP, il est nécessaire de décrire la fonction R de récompense. Une représentation similaire à la description de la fonction de transition peut être utilisée pour le cadre des FMDP. En effet, la fonction de récompense d'un MDP peut, d'une part, être décomposée de façon additive et, d'autre part, ne dépend pas nécessairement de toutes les variables d'état du problème.

Par exemple, dans le problème *Coffee Robot*, la fonction de récompense, représentée par un losange dans la figure 1.2, ne dépend que des variables $\mathcal{HOC}$ et $\mathcal{W}$ et elle est indépendante des actions réalisées par le robot ou bien des autres variables du problème.



(a)

$\mathcal{HOC}$	$\mathcal{W}$	R
1	0	1.0
1	1	0.9
0	0	0.1
0	1	0.0

(b)

$\mathcal{HOC}$	R_0
0	0.0
1	0.9

$+$

$\mathcal{W}$	R_1
0	0.1
1	0.0

(c)

Figure 1.2. *Représentation de la fonction de récompense $R(s)$*

Le tableau de la figure 1.2(b) spécifie que le meilleur état pour le robot est lorsque sa propriétaire a un café et que le robot est sec tandis que le pire cas est lorsque sa propriétaire n'a pas de café et que le robot est mouillé. On observe la préférence donnée au cas où l'utilisateur possède un café et le robot est mouillé par rapport au cas où l'utilisateur n'a pas de café et le robot est sec.

[BOU 00] définissent la fonction de récompense du problème *Coffee Robot* en faisant la somme des deux critères du problème, « la propriétaire a un café » et « le robot est mouillé ». Pourtant, ces deux critères sont indépendants. Afin de profiter de la décomposition additive de cette fonction de récompense, [GUE 03b] proposent de formaliser la fonction de récompense d'un FMDP en une somme de plusieurs *fonctions de récompense localisées* [3].

Pour le problème *Coffee Robot*, on peut définir la fonction de récompense comme la somme de deux fonctions de récompenses localisées : la première associée à la variable $\mathcal{HOC}$ et la deuxième associée à la variable $\mathcal{W}$ et représentant respectivement les deux critères « la propriétaire a un café » et « le robot est mouillé ».

[GUE 03b] formalisent cette notion en définissant tout d'abord la notion de *scope* d'une fonction f localisée (notée $\text{SCOPE}(f)$), que nous traduirons par « portée ». La portée d'une fonction f localisée est définie telle que :

DÉFINITION 1.1.– *scope*
Soit une fonction $f : \{X_1, \ldots, X_n\} \mapsto \mathbb{R}$. $\text{SCOPE}(f) = C$ *définit la portée de f si et seulement si f ne dépend que des variables de C, les valeurs des autres variables étant indifférentes. On assimilera alors f à sa projection sur* $\text{DOM}(C)$ *: $f : \text{DOM}(C) \mapsto \mathbb{R}$ avec $C \subseteq \{X_1, \ldots, X_n\}$.*

Soit une fonction f telle que $\text{SCOPE}(f) = C$ avec $C \subseteq X$, on utilise la notation $f(x)$ comme raccourci pour noter $f(x[C])$ avec $x[C]$ représentant l'instanciation des

3. *Localized reward functions.*

variables appartenant à C dans l'instanciation x. La portée d'une fonction f permet ainsi de mettre en évidence les indépendances relatives à f[4].

Il est maintenant possible de définir le concept de fonction de récompense localisée. Soit un ensemble de fonctions localisées $R_1^a, \ldots, R_r^a$ avec la portée de chaque fonction R_i^a restreinte à un sous-ensemble $C_i^a \subseteq \{X_1, \ldots, X_n\}$. La récompense associée au fait d'exécuter l'action a dans un état s est alors définie telle que :

$$R^a(s) = \sum_{i=1}^{r} R_i^a(s[C_i^a]) \tag{1.2}$$

$$= \sum_{i=1}^{r} R_i^a(s). \tag{1.3}$$

Ainsi, pour reprendre l'exemple de *Coffee Robot*, le problème est défini par deux fonctions de récompenses R_1 et R_2 définies dans la figure 1.2(c) et correspondant respectivement aux deux critères « la propriétaire a un café » et « le robot est mouillé ». On a $\text{SCOPE}(R_1) = \{\mathcal{HOC}\}$ et $\text{SCOPE}(R_2) = \{\mathcal{W}\}$. On utilise $R_1(s)$ comme raccourci pour représenter $R_1(s[\mathcal{HOC}])$, avec $s[\mathcal{HOC}]$ représentant l'instanciation de $\mathcal{HOC}$ dans s.

Bien que l'ensemble des algorithmes décrits dans le cadre des FMDP exploitent les indépendances relatives aux fonctions de récompense du problème, tous n'exploitent pas la décomposition additive de la fonction de récompense. De plus, tous les problèmes ne présentent pas une telle décomposition.

1.2.4. *Indépendances relatives aux contextes*

Les indépendances relatives aux contextes concernent le fait que, pour représenter une fonction du problème à résoudre (quelle que soit la fonction), il n'est pas obligatoire de tester systématiquement l'ensemble des variables nécessaires à la représentation de cette fonction. Un contexte se définit de la façon suivante :

DÉFINITION 1.2.– *Contexte*
Soit une fonction $f : \{X_0, \ldots, X_n\} \rightarrow Y$. Un contexte $c \in \text{DOM}(C)$ est une instanciation d'un sous-ensemble de variables $C = \{C_0, \ldots, C_j\}$ tel que $C \subseteq \{X_0, ..., X_n\}$. Un contexte est noté $(C_0 = c_0) \wedge \ldots \wedge (C_j = c_j)$ ou $C_0 = c_0 \wedge \ldots \wedge C_j = c_j$.

4. La notion de portée d'une fonction est similaire à la notion de parent utilisée pour la définition des distributions de probabilité conditionnelles de la fonction de transition.

Par exemple, la description de la politique optimale dans le problème *Coffee Robot* nécessite l'utilisation de toutes les variables du problème. Cependant, dans le contexte $\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 1 \wedge \mathcal{O} = 1$, c'est-à-dire lorsque la propriétaire n'a pas de café et que le robot a un café et qu'il est au bureau, il est possible de déterminer l'action optimale (l'action $\mathcal{D}\mathrm{elC}$ dans ce cas) sans avoir à tester d'autres variables telles que « est-ce qu'il pleut ? » ou « le robot est-il mouillé ? ».

Contrairement aux indépendances fonctionnelles, l'exploitation des indépendances contextuelles est étroitement liée aux structures de données employées pour représenter les fonctions du problème. En effet, les opérateurs, PARENTS et SCOPE, qui s'appliquent aux indépendances fonctionnelles définissent le nombre de variables dont une fonction dépend. Comme nous l'avons constaté dans le paragraphe 1.2.3 (par exemple dans la figure 1.1), la spécification des indépendances fonctionnelles permet de représenter ces fonctions de façon plus compacte, même lorsque la structure de données utilisée pour leur représentation n'est pas structurée, comme c'est le cas pour les représentations tabulaires.

Pour un ensemble de variables donné (spécifiées par les opérateurs PARENTS et SCOPE), les indépendances contextuelles sont exploitées pour représenter des fonctions de façon plus compacte. Pour ces indépendances, la principale technique est d'utiliser des représentations structurées permettant d'agréger des états, contrairement à une représentation tabulaire.

Ainsi, chaque algorithme utilisant une structure de données différente, nous avons préféré illustrer ce concept dans la section suivante, c'est-à-dire dans le cadre de l'algorithme qui l'exploite et des structures de données utilisées.

1.3. Planification dans les FMDP

La section suivante présente un certain nombre de méthodes de planification dans les FMDP. Plutôt que de décrire les algorithmes en détails, nous nous sommes attachés à décrire les différentes représentations utilisées par ceux-ci, afin que le lecteur puisse distinguer rapidement les principales différences et caractéristiques de ces algorithmes. Cependant, nous donnerons l'ensemble des références nécessaires pour que le lecteur puisse obtenir une description exhaustive de ces algorithmes s'il le désire.

1.3.1. *Itérations structurées sur les valeurs et sur les politiques*

Les deux algorithmes d'itération structurée sur les valeurs et sur les politiques, *Structured Value Iteration* (SVI) et *Structured Policy Iteration* (SPI) [BOU 00] ont été les premiers algorithmes adaptant les algorithmes de programmation dynamique au

formalisme des FMDP, illustrant ainsi les avantages (et les inconvénients) de ce formalisme. En plus des indépendances spécifiques aux fonctions utilisées dans la décomposition des fonctions de transition et de récompense, les algorithmes SPI et SVI utilisent une représentation structurée afin d'exploiter les indépendances contextuelles dans la représentation des différentes fonctions du problème.

Par exemple, nous pouvons constater que, dans l'exemple *Coffee Robot*, lorsque la propriétaire a déjà un café, alors il n'est pas nécessaire d'évaluer si le robot a un café ou s'il est au bureau pour déterminer si la propriétaire aura un café au prochain pas de temps. Ainsi, la distribution de probabilité de la variable aléatoire $\mathcal{HOC}'$ dans le contexte $\mathcal{HOC} = 1$, c'est-à-dire « la propriétaire a un café », est indépendante des variables $\mathcal{HRC}$ et $\mathcal{O}$ au pas de temps précédent, c'est-à-dire « le robot a-t-il un café ? » et « le robot est-il au bureau ? », bien que ces deux variables soient nécessaires pour définir complètement la distribution de probabilité de $\mathcal{HOC}'$.

Pour exploiter ce type de régularités, [BOU 00] suggèrent plusieurs notations pour représenter les fonctions du FMDP à résoudre, telles que les règles [POO 97], les listes de décision [RIV 87] ou les diagrammes de décision binaires [BRY 86]. SPI et SVI sont présentés en utilisant les arbres de décision [QUI 93], principalement à cause de leur simplicité. Nous verrons aussi deux autres méthodes de résolution dans les FMDP utilisant d'autres représentations (paragraphes 1.3.2 et 1.3.3).

1.3.1.1. *Les arbres de décision*

Les arbres de décision représentent une fonction en partitionnant son espace d'entrée. Un arbre de décision est composé de :

– *nœuds intérieurs* (ou nœuds de décision) : ils représentent un test sur une variable de l'espace d'entrée. Ils sont parents d'autres nœuds dans l'arbre et définissent la structure de la partition de l'espace d'entrée ;

– *branches* : elles connectent un nœud intérieur parent à un nœud enfant en restreignant le domaine des valeurs de la variable en fonction d'un test installé au nœud intérieur parent ;

– *feuilles* : elles représentent les nœuds terminaux de l'arbre et sont associées à la valeur de la fonction dans la partition définie par l'ensemble des tests des nœuds intérieurs qui sont les parents de la feuille.

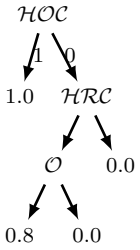
Dans le cadre de SPI et SVI, les arbres de décision sont utilisés pour représenter l'ensemble des fonctions du FMDP à résoudre. Une fonction F représentée avec un arbre de décision est notée $\mathsf{Tree}\,[F]$. Graphiquement, les arbres sont représentés en utilisant la convention suivante : pour un nœud de décision testant une variable X booléenne, les branches de gauche et de droite sont associées respectivement à $X = 1$ et $X = 0$. Lorsque la variable n'est pas booléenne, alors la valeur de X est indiquée sur chaque branche.

1.3.1.2. *Représentation de la fonction de transition*

Dans le problème *Coffee Robot*, la description sous forme tabulaire de la distribution de probabilité $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O},\mathcal{HRC},\mathcal{HOC})$, rappelée figure 1.3(a), fait apparaître plusieurs régularités pouvant être agrégées. Par exemple, on peut remarquer que, comme décrit ci-dessus, dans le contexte $\mathcal{HOC} = 1$, la probabilité que $\mathcal{HOC}'$ soit vrai est égale à 1 quelle que soit la valeur des deux autres variables $\mathcal{O}$ et $\mathcal{HRC}$ appartenant à l'ensemble $\textsc{Parents}_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}')$. En effet, lorsque la propriétaire a un café, alors il est certain qu'elle aura un café au prochain pas de temps. Les arbres de décision permettent de représenter de façon compacte ce type de régularités.

$\mathcal{HOC}$	$\mathcal{HRC}$	$\mathcal{O}$	$\mathcal{HOC}'$
1	1	1	1.0
1	1	0	1.0
1	0	1	1.0
1	0	0	1.0
0	1	1	0.8
0	1	0	0.0
0	0	1	0.0
0	0	0	0.0

(a)



(b)

Figure 1.3. *Représentation sous la forme tabulaire de la distribution de probabilité conditionnelle $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O},\mathcal{HRC},\mathcal{HOC})$ (figure a) et sous la forme d'un arbre de décision (figure b). La feuille notée* 0.8 *signifie que la probabilité pour la variable $\mathcal{HOC}'$ d'être vraie est : $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O}=1,\mathcal{HRC}=1,\mathcal{HOC}=0) = 0.8$. Ainsi, certaines régularités sont agrégées, comme par exemple les probabilités $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{HOC}=1) = 1.0$.*

Un arbre de décision Tree $[P_\tau(X'|\textsc{Parents}_\tau(X'))]$ représentant la distribution de probabilité conditionnelle $P_\tau(X'|\textsc{Parents}_\tau(X'))$ est composé de :

– *nœuds intérieurs* : représentent un test sur une variable $X_j \in \textsc{Parents}_\tau(X')$;

– *branches* : représentent une valeur $x_j \in \textsc{Dom}(X_j)$ de la variable X_j testée au nœud parent et définissant le sous-espace représenté par le nœud enfant connecté à la branche ;

– *feuilles* : représentent la distribution de probabilité $P_f(X'|x[X_j])$, avec $x[X_j]$ l'ensemble des instanciations des variables $X_j \in \textsc{Parents}_\tau(X')$ testées dans les nœuds parents de la feuille f dans l'arbre.

L'interprétation d'un tel arbre est directe : la distribution de probabilité d'une variable X' pour une instanciation x est donnée par l'unique feuille que l'on atteint en choisissant à chaque nœud de décision la branche correspondant à une valeur de test cohérente avec x. Le chemin en question peut spécifier une instanciation partielle de X'.

La figure 1.3(b) représente Tree $\left[P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O},\mathcal{HRC},\mathcal{HOC})\right]$: la distribution de probabilité conditionnelle $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}'|\mathcal{O},\mathcal{HRC},\mathcal{HOC})$ sous la forme d'un arbre de décision. Les valeurs aux feuilles indiquent la probabilité que la variable $\mathcal{HOC}'$ soit vraie. On peut alors remarquer qu'une représentation en arbre de décision, pour la définition de $P_{\mathcal{D}\mathrm{elC}}(\mathcal{HOC}')$, est plus compacte qu'une représentation tabulaire puisqu'elle exploite les indépendances contextuelles : 4 feuilles sont nécessaires à la représentation de la fonction alors que 8 entrées sont nécessaires pour décrire la même fonction sous forme tabulaire. Nous verrons que cette factorisation est utilisée par les algorithmes de planification SPI et SVI.

1.3.1.3. *Représentation de la fonction de récompense*

La représentation d'une fonction de récompense avec des arbres de décision est très similaire à la représentation d'une distribution de probabilité. En effet, la signification des nœuds intérieurs et des branches est la même. Seule change l'étiquette attachée aux feuilles de l'arbre puisqu'elle représente des nombres réels plutôt que des distributions de probabilité.

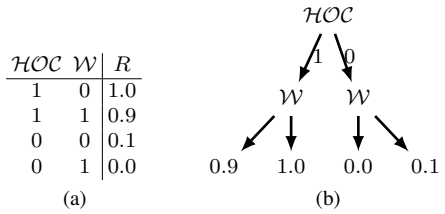


Figure 1.4. *Définition de la fonction de récompense $R(s)$ avec une représentation tabulaire (figure a) et un arbre de décision (figure b). La feuille notée 0.9 signifie $R(\mathcal{HOC}=1,\mathcal{W}=1)=0.9$.*

La figure 1.4 représente la fonction de récompense pour le problème *Coffee Robot* et compare la représentation tabulaire $R(s)$ avec l'arbre de décision Tree $[R(s)]$. On constate qu'aucune indépendance contextuelle n'est utilisable puisque le nombre de feuilles dans l'arbre est égal au nombre de lignes nécessaires à la définition de la fonction avec une représentation tabulaire.

Les algorithmes SPI et SVI ne permettent pas d'exploiter la décomposition additive d'une fonction de récompense telle qu'elle a été définie dans le paragraphe 1.2.3.4.

1.3.1.4. *Représentation d'une politique*

Une politique $\pi(s)$ peut aussi être représentée sous la forme d'un arbre de décision Tree $[\pi(s)]$. La figure 1.5 représente une politique stationnaire déterministe Tree $[\pi(s)]$ dans le problème *Coffee Robot*.

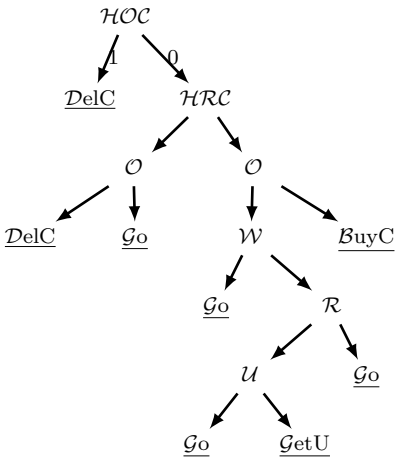


Figure 1.5. *Représentation d'une politique $\pi(s)$ sous la forme d'un arbre de décision* $\mathsf{Tree}\,[\pi(s)]$. *La feuille notée* $\underline{\mathcal{B}\mathrm{uyC}}$ *signifie* $\pi(\mathcal{HOC}=0,\mathcal{HRC}=0,\mathcal{O}=0)=\underline{\mathcal{B}\mathrm{uyC}}$.

L'espace d'état du problème *Coffee Robot* est composé de 6 variables binaires. Une description tabulaire de π aurait donc nécessité $2^6 = 64$ entrées. L'arbre $\mathsf{Tree}\,[\pi]$ ne contient que 8 feuilles (15 nœuds au total). Sur le problème *Coffee Robot*, l'utilisation d'arbres de décision pour représenter une politique permet donc d'exploiter des indépendances contextuelles telles que, lorsque la propriétaire n'a pas de café, que le robot est au bureau et qu'il a un café, il n'est pas nécessaire de connaître la valeur des variables telles que « est-ce qu'il pleut ? » pour déterminer la meilleure action à réaliser.

Lors de l'exécution d'une politique dans un environnement, une telle représentation se révèle avantageuse lorsque déterminer la valeur d'une variable a un coût (par exemple en termes de temps de calcul). En effet, elle permet de n'avoir à déterminer que la valeur des variables strictement nécessaires à l'exécution de la politique de façon spécifique à l'état courant de l'agent. Une telle propriété permet ainsi d'économiser l'évaluation des variables inutiles.

Enfin, l'utilisation d'un arbre de décision pour la description d'une politique permet d'effectuer un nombre réduit de tests pour déterminer l'action à réaliser pour l'agent. Dans le pire cas, pour un problème décrit avec N variables, seuls N tests sont nécessaires pour déterminer l'action retournée par la politique. Cependant, l'espace mémoire requis pour une telle représentation reste, dans le pire cas, exponentielle en fonction du nombre de variables décrivant l'espace d'états du problème.

1.3.1.5. *Représentation d'une fonction de valeur*

Naturellement, la fonction de valeur V_π d'une politique π peut aussi se représenter sous la forme d'un arbre de décision $\mathsf{Tree}\,[V_\pi]$. La sémantique d'un tel arbre est quasiment identique à celle d'un arbre de décision représentant une fonction de récompense : un nœud de décision représente une variable, une branche représente la valeur de la variable testée au nœud de décision parent et les feuilles représentent la valeur de la fonction de valeur pour la partition délimitée par les tests de ses parents. La figure 1.6 représente la fonction de valeur de la politique $\mathsf{Tree}\,[\pi]$ représentée figure 1.5.

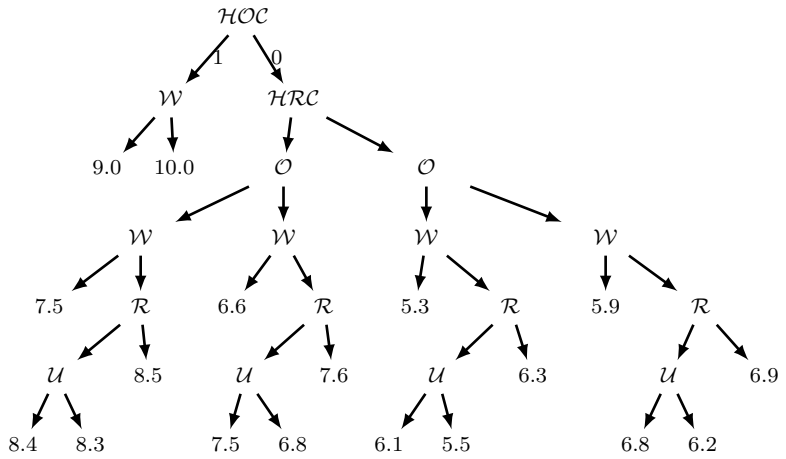


Figure 1.6. *Représentation de la fonction de valeur $V_\pi(s)$ de la politique π sous la forme d'un arbre de décision* $\mathsf{Tree}\,[V_\pi(s)]$ *pour le problème* Coffee Robot. *La feuille notée* 10.0 *signifie* $V_\pi(\mathcal{HOC}=1,\mathcal{W}=0)=10.0$.

L'arbre $\mathsf{Tree}\,[V_\pi]$ ne contient que 18 feuilles (35 nœuds au total) alors qu'une représentation tabulaire aurait nécessité 64 entrées. Sur le problème *Coffee Robot*, une représentation sous la forme d'un arbre de décision permet donc d'exploiter les indépendances contextuelles. Par exemple, la valeur $V_\pi(\mathcal{HOC}=1,\mathcal{W}=0)$ de la politique π, lorsque la propriétaire a un café et que le robot est sec, ne dépend pas des autres variables du problème. Une telle propriété peut être considérée comme l'agrégation de plusieurs états. Ainsi, lors du calcul itératif d'une fonction de valeur, il n'est nécessaire de calculer qu'une seule fois la mise à jour de la valeur d'une feuille pour mettre à jour la valeur de tous les états représentés par cette feuille.

Cependant, il est possible de constater sur la fonction de valeur $\mathsf{Tree}\,[V_\pi]$ qu'une telle représentation ne permet pas d'exploiter certaines régularités présentes dans la

définition de V_π. En effet, on peut remarquer, par exemple, que la structure des sous-arbres composés des variables $\mathcal{R}$, $\mathcal{W}$, $\mathcal{U}$ et $\mathcal{O}$ est identique. Nous verrons qu'une approximation additive de la fonction de valeur (que nous présenterons paragraphe 1.3.3.5) permet d'exploiter une telle symétrie, contrairement à une représentation telle que les arbres de décision.

Enfin, dans le pire cas, c'est-à-dire lorsque la fonction de valeur de la politique évaluée est différente pour tous les états possibles, la taille de la représentation augmente exponentiellement avec le nombre de variables composant l'espace d'états du problème.

1.3.1.6. *Algorithmes*

Le principe de base des algorithmes SPI et SVI est d'adapter les algorithmes *Policy Iteration* et *Value Iteration* aux arbres de décision. Ainsi, plutôt que d'avoir à calculer une mise à jour de la valeur de chaque état possible lors d'une itération, comme c'est le cas pour *Policy Iteration* et *Value Iteration*, SVI et SPI calculent cette mise à jour pour chaque feuille d'un arbre de décision, permettant de réduire le coût des calculs lorsque plusieurs états sont agrégés et représentés par la même feuille. [BOU 00] propose une description exhaustive de ces deux algorithmes, que nous ne rappelons pas ici puisque nous nous concentrons sur les représentations.

1.3.2. *L'algorithme* Stochastic Planning Using Decision Diagrams

Dans certains problèmes, la fonction de valeur possède des symétries qui ne sont pas exploitées par les arbres de décision, notamment lorsque la fonction est strictement identique dans plusieurs contextes disjoints. L'algorithme présenté par [HOE 99], nommé SPUDD[5], propose d'utiliser des diagrammes de décision algébriques[6] (ADD), décrits par [BAH 93], pour représenter les fonctions d'un FMDP. De façon semblable à SPI, SPUDD exploite les indépendances relatives à la fois aux fonctions et aux contextes.

L'utilisation d'ADD plutôt que d'arbres de décision présente deux avantages supplémentaires. D'une part, les ADD permettent de mieux factoriser certaines fonctions en exploitant le fait que certaines sous-parties d'une partition de l'espace sont semblables les unes aux autres, alors que les contextes les caractérisant sont disjoints. D'autre part, les variables utilisées dans un ADD sont ordonnées. Bien que trouver un ordre optimal des variables à tester pour représenter une fonction de façon la plus compacte possible est un problème difficile, [HOE 00] montrent que plusieurs heuristiques peuvent être utilisées pour trouver un ordre permettant de représenter les fonctions de

5. *Stochastic Planning Using Decision Diagrams.* (Planification stochastique utilisant les diagrammes de décision).

6. *Algebraic Decision Diagrams.*

façon suffisamment compacte pour accélérer nettement les calculs. Un tel ordonnancement est utilisé pour accélérer les calculs réalisés sur les fonctions représentées. Ces deux avantages permettent d'améliorer les algorithmes de programmation dynamique aussi bien en termes d'espace mémoire consommé qu'en termes de temps de calcul.

1.3.2.1. *Les diagrammes de décision algébriques*

Les ADD sont une généralisation des diagrammes de décision binaires (BDD) ou *Binary Decision Diagrams* [BRY 86]. Les BDD sont une représentation compacte de fonctions $\mathcal{B}^n \rightarrow \mathcal{B}$ de n variables binaires vers une valeur binaire. Les ADD généralisent les BDD pour représenter des fonctions réelles $\mathcal{B}^n \rightarrow \mathbb{R}$ de n variables binaires vers une valeur réelle. Un ADD est composé de :

– *nœuds intérieurs* (ou nœuds de décision) : ils représentent un test sur une variable binaire de l'espace d'entrée. Ils sont le parent de deux branches correspondant respectivement au fait que la variable testée est égale à `Vrai` ou `Faux` ;

– *branches* : elles connectent un nœud intérieur parent à un nœud enfant en fonction de la valeur `Vrai` ou `Faux` du test installé au nœud intérieur ;

– *feuilles* : elles représentent les nœuds terminaux du diagramme et sont associées à la valeur de la fonction dans l'un des sous-espaces définis par l'ensemble des tests des nœuds intérieurs parents de la feuille.

Contrairement à un arbre de décision, les nœuds intérieurs et les feuilles d'un ADD peuvent avoir plusieurs parents. Une fonction F représentée avec un ADD est notée $\mathsf{ADD}\,[F]$. La convention suivante est utilisée pour représenter un ADD graphiquement : pour un nœud de décision testant une variable X, les branches dessinées en trait plein et pointillé sont associées respectivement à $X = 1$ et $X = 0$.

Les ADD possèdent plusieurs propriétés intéressantes. D'une part, pour un ordre de variables donné, chaque fonction distincte n'a qu'une seule représentation. D'autre part, la taille de la représentation de nombreuses fonctions peut être réduite grâce à la réutilisation de sous-graphes identiques au sein de la description. Enfin, il existe des algorithmes optimisés pour la plupart des opérations de base, notamment la multiplication, l'addition ou bien la maximisation.

La figure 1.7 montre l'exemple d'une même fonction F représentée par un arbre de décision et par un ADD. Elle illustre le fait que les arbres de décision, contrairement aux ADD, ne sont pas adaptés pour la représentation de certaines fonctions, notamment les fonctions disjonctives. Ainsi, alors que la représentation $\mathsf{Tree}\,[F]$ contient 5 feuilles différentes (et 4 nœuds intérieurs), la représentation $\mathsf{ADD}\,[F]$ n'en contient que 2 (plus 3 nœuds intérieurs). La mise à jour de cette fonction dans le cas de SPI nécessitera donc 5 calculs de mise à jour différents alors que SPUDD ne réalisera que 2 calculs.

Cependant, l'utilisation des ADD impose principalement deux contraintes sur le FMDP à résoudre. Premièrement, il est nécessaire que les variables du FMDP soient

toutes binaires, les ADD ne représentant que des fonctions $\mathcal{B}^n \rightarrow \mathbb{R}$. Pour les problèmes contenant des variables à plus de deux valeurs, il est toujours possible de décomposer ces variables avec de nouvelles variables (binaires). Deuxièmement, les algorithmes basés sur les ADD supposent que, au sein de la structure de données, les tests sur les variables sont ordonnés. Lorsque ces deux contraintes sont satisfaites, il est possible de représenter l'ensemble des fonctions du FMDP à résoudre en utilisant des ADD.

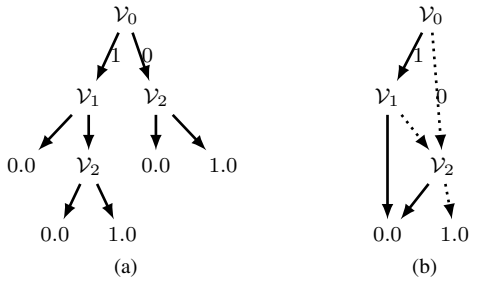


Figure 1.7. *Comparaison des représentations d'une fonction F sous la forme d'un arbre de décision* Tree $[F]$ *(figure a) et d'un diagramme de décision algébrique* ADD $[F]$ *(figure b).*

De la même façon que pour SPI et SVI, la plupart des opérateurs sur les fonctions sont redéfinis et optimisés pour manipuler des ADD. L'algorithme SPUDD reprend le principe de l'algorithme *Value Iteration* pour l'adapter aux ADD en supposant que toutes les variables du FMDP à résoudre sont binaires et que les variables sont préalablement ordonnées.

Les travaux sur SPUDD ont été prolongés avec APRICODD [STA 01] qui est une implémentation de SPUDD avec plusieurs améliorations. Premièrement, plusieurs étapes du calcul de l'équation de Bellman sont optimisées afin de permettre à l'utilisateur de pouvoir paramétrer un compromis entre temps de calcul et espace mémoire nécessaire. De plus, il est possible de calculer des fonctions de valeur approchées en spécifiant soit une taille maximale de l'ADD représentant la fonction de valeur, soit une erreur maximale de la représentation [HOE 00]. Enfin, APRICODD propose plusieurs méthodes de réorganisation automatique des variables afin d'éviter à l'utilisateur d'avoir à les spécifier manuellement. La dernière version d'APRICODD est disponible sur Internet[7]. Les résultats présentés dans [HOE 99, STA 01] suggèrent qu'une telle approche est plus efficace que celle utilisée par les algorithmes SPI ou SVI.

7. `http://www.cs.toronto.edu/~jhoey/spudd`

1.3.3. *Programmation linéaire approchée dans un FMDP*

Une alternative à la programmation dynamique pour résoudre un MDP est l'utilisation de la programmation linéaire (paragraphe **??**, volume 1). L'utilisation de cette technique pour la résolution d'un FMDP est l'aboutissement de nombreux travaux commencés par [KOL 99, KOL 00] puis menés principalement par Guestrin [GUE 01, GUE 03a, GUE 03b].

La fonction de valeur optimale d'un MDP peut être calculée en formulant celui-ci sous la forme d'un programme linéaire [MAN 60] :

$$\begin{array}{ll} \text{Pour les variables :} & V(s), \forall s \in S \ ; \\ \text{Minimiser :} & \sum_s \alpha(s) V(s) \ ; \\ \text{Avec les contraintes :} & V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \\ & \forall s \in S, \forall a \in A. \end{array} \qquad \text{(LP 1)}$$

où $\alpha(s) > 0$ est la pondération d'intérêt de l'état s. La résolution de ce programme linéaire se heurte à un problème de complexité à la fois dans la fonction à optimiser, les variables à déterminer et le nombre de contraintes, ce qui impose d'avoir recours à une méthode approchée pour traiter des problèmes de grande taille.

Ces problèmes sont résolus en exploitant deux idées principales reposant principalement sur les indépendances fonctionnelles et la décomposition additive de la fonction de récompense.

La première idée exploite une représentation approchée de la fonction de valeur, plus précisément une combinaison linéaire de fonctions de base [SCH 85], pour, d'une part, diminuer la complexité de la définition de la fonction à optimiser et du nombre de variables à déterminer et pour, d'autre part, accélérer le calcul de la génération des contraintes. La deuxième idée propose d'utiliser un algorithme de décomposition des contraintes afin de pouvoir représenter l'ensemble des contraintes du programme linéaire de façon compacte.

Ces deux idées sont exploitées par deux algorithmes différents proposés par [GUE 03b]. Le premier algorithme est une reformulation de l'algorithme *Policy Iteration* utilisant la programmation linéaire pour la phase d'évaluation de la politique. Le deuxième algorithme part directement du programme linéaire LP 1 et propose la construction directe d'un programme linéaire afin d'évaluer la fonction de valeur optimale du FMDP à résoudre. La section suivante présente les représentations utilisées par ces deux algorithmes.

1.3.3.1. *Représentations*

Principalement deux représentations sont utilisées dans l'utilisation de la programmation linéaire telle qu'elle est proposée par Guestrin. La première représentation est

une représentation tabulaire classique et permet d'exploiter uniquement les propriétés d'indépendance fonctionnelle et de décomposition additive du problème. La deuxième représentation est une représentation structurée basée sur des règles [ZHA 99] permettant en plus d'utiliser les indépendances contextuelles au sein d'une fonction. Bien que [GUE 03b] montrent que, pour certains problèmes, une représentation tabulaire est plus rapide qu'une représentation structurée, nous pensons que les représentations structurées sont mieux adaptées pour représenter des problèmes réels, justement parce qu'elles exploitent les indépendances contextuelles. De plus, le pire des cas des représentations structurées est souvent moins mauvais que le pire des cas des représentations tabulaires en termes de temps de calcul [STA 01, GUE 03a].

Deux avantages sont avancés par [GUE 03b] pour justifier l'utilisation des règles plutôt qu'une autre représentation telle que les arbres de décision ou les ADD. Premièrement, cette représentation est bien adaptée à leur technique de décomposition des contraintes du programme linéaire. Deuxièmement, contrairement aux arbres de décision ou aux ADD, les règles utilisées pour décrire une fonction peuvent ne pas être exclusives. Deux types de règles sont distinguées : les règles de probabilité (*probability rules*) et les règles de valeur (*value rules*). Les règles de probabilité sont utilisées pour représenter la fonction de transition alors que les règles de valeur sont utilisées pour définir les fonctions de récompense ainsi que les fonctions de valeur. Ces deux types de règles et leurs utilisations dans le cadre de la programmation linéaire approchée dans un FMDP sont décrits dans la suite de cette section, d'après [GUE 03b]. Une fonction $F(x)$ représentée avec un ensemble de règles est notée $\mathsf{Rule}[F]$.

1.3.3.2. *Représentation de la fonction de transition*

Le premier type de règles est utilisé pour représenter la fonction de transition, plus précisément les distributions de probabilité conditionnelles quantifiant les DBN. Une règle correspond à un ou plusieurs contextes dans la distribution ayant la même probabilité. Nous commençons par définir la cohérence entre deux contextes :

DÉFINITION 1.3.– *Cohérence entre deux contextes*
Soit $C \subseteq \{X, X'\}$, $c \in \text{DOM}(C)$, $B \subseteq \{X, X'\}$ et $b \in \text{DOM}(B)$. On dit que les deux contextes b et c sont cohérents s'ils ont tous les deux les mêmes valeurs pour toutes les variables appartenant à l'intersection $C \cap B$.

Ainsi, des contextes possédant des variables avec des valeurs identiques sont définis comme étant *cohérents*. Les probabilités ayant la même valeur et des contextes cohérents sont représentées avec des règles de probabilité :

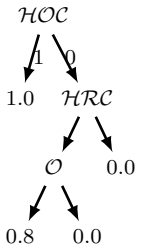
DÉFINITION 1.4.– *Règle de probabilité*
Une règle de probabilité $\eta = |c : p|$ est une fonction $\eta : \{X, X'\} \mapsto [0, 1]$ avec le contexte $c \in \text{DOM}(C)$, $C \subseteq \{X, X'\}$ et $p \in [0, 1]$ et tel que $\eta(s, x') = p$ si les instanciations s et x' sont cohérentes avec c, ou sinon est égal à 1.

Deux règles sont dites cohérentes si leurs contextes respectifs sont cohérents. On définit maintenant un ensemble de règles de probabilité pour définir complètement une distribution de probabilité conditionnelle :

DÉFINITION 1.5.– *Ensemble de règles de probabilité*
Un ensemble de règles P_a d'une distribution de probabilité conditionnelle est une fonction $P_a : (\{X'_i\} \cup X) \mapsto [0,1]$ composée des règles de probabilité $\{\eta_1, \ldots, \eta_m\}$ dont les contextes sont mutuellement exclusifs et exhaustifs. On définit : $P_a(x'_i|x) = \eta_j(x, x'_i)$ avec η_j l'unique règle appartenant à P_a dont le contexte c_j est cohérent avec (x'_i, x). De plus, on a nécessairement : $\forall s \in S : \sum_{x'_i} P_a(x'_i|s) = 1$.

Il est possible de définir $\text{PARENTS}_a(X'_i)$ comme l'union des variables appartenant aux contextes des règles appartenant à $P_a(X'_i)$.

A l'instar des arbres de décision, les ensembles de règles de probabilité permettent d'exploiter les indépendances contextuelles. De plus, les arbres de décision forment une partition complète d'un espace. Il est donc facile de définir un ensemble de règles mutuellement exclusives et exhaustives à partir d'un arbre de décision, comme le montre la figure 1.8 pour définir $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}')$.



(a)

$$\left|\begin{array}{ll}
\mathcal{HOC} = 1 \wedge \mathcal{HOC}' = 1 & : 1.0 \\
\mathcal{HOC} = 0 \wedge \mathcal{O} = 0 \wedge \mathcal{HOC}' = 0 & : 1.0 \\
\mathcal{HOC} = 0 \wedge \mathcal{O} = 1 \wedge \mathcal{HRC} = 1 \wedge \mathcal{HOC}' = 0 & : 0.2 \\
\mathcal{HOC} = 0 \wedge \mathcal{O} = 1 \wedge \mathcal{HRC} = 1 \wedge \mathcal{HOC}' = 1 & : 0.8 \\
\mathcal{HOC} = 0 \wedge \mathcal{O} = 1 \wedge \mathcal{HRC} = 0 \wedge \mathcal{HOC}' = 0 & : 1.0
\end{array}\right|$$

(b)

Figure 1.8. *Représentation de la distribution de probabilité conditionnelle $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}')$ sous la forme d'un arbre de décision et d'un ensemble de règles. La règle $|\mathcal{HOC} = 0 \wedge \mathcal{O} = 1 \wedge \mathcal{HRC} = 1 \wedge \mathcal{HOC}' = 1 : 0.8|$ définit $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}' = 1|\mathcal{HOC} = 0, \mathcal{O} = 1, \mathcal{HRC} = 1) = 0.8$.*

La probabilité $P_{\mathcal{D}\text{elC}}(\mathcal{HOC}' = 1|\mathcal{HOC} = 0, \mathcal{O} = 1, \mathcal{HRC} = 1) = 0.8$ est représentée par la règle correspondante $|\mathcal{HOC} = 0 \wedge \mathcal{O} = 1 \wedge \mathcal{HRC} = 1 \wedge \mathcal{HOC}' = 1 : 0.8|$. On peut remarquer que, pour les tests concernant les variables X au temps t, le contexte de cette règle correspond aux tests réalisés dans l'arbre de décision pour atteindre la feuille 0.8. De plus, la variable X'_i au temps $t+1$ appartiennent aussi au contexte de la règle. Une distribution de probabilité conditionnelle $F(x)$ représentée avec un ensemble de règles de probabilité est notée $\mathsf{Rule}_\mathsf{p}[F]$.

1.3.3.3. *Représentation de la fonction de récompense*

Pour représenter la fonction de récompense d'un FMDP, on définit les règles de valeur :

DÉFINITION 1.6.– *Règle de valeur*
Une règle de valeur $\rho = |c : v|$ *est une fonction* $\rho : X \rightarrow \mathbb{R}$ *telle que* $\rho(x) = v$ *lorsque* x *est cohérent avec le contexte* c *et 0 sinon.*

On note que la portée d'une règle de valeur est $\text{SCOPE}(\rho) = C$ avec C l'ensemble des variables instanciées dans le contexte c de la règle $\rho = |c : v|$.

Il est maintenant possible de définir une fonction comme un ensemble de règles de valeur :

DÉFINITION 1.7.– *Ensemble de règles de valeur*
Un ensemble de règles de valeur représentant une fonction $f : X \mapsto \mathbb{R}$ *est composé de l'ensemble des règles de valeur* $\{\rho_1, \ldots, \rho_n\}$ *telles que* $f(x) = \sum_{i=1}^{n} \rho_i(x)$ *avec* $\forall i : \text{SCOPE}(\rho_i) \subseteq X$.

Une fonction F représentée avec un ensemble de règles de valeur est notée $\mathsf{Rule_v}[F]$. De plus, on suppose qu'une récompense $R(s, a)$ peut s'écrire sous la forme d'une somme de fonctions de récompense dont la portée est limitée :

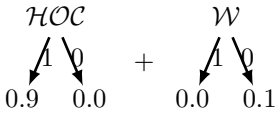
$$R(s, a) = \sum_j r_j^a(s). \tag{1.4}$$

Cette représentation permet de représenter de façon naturelle des fonctions en exploitant à la fois des indépendances contextuelles et une décomposition additive, comme le montre la figure 1.9.

Représentation tabulaire :

$$R(s) = \begin{array}{cc|c} \mathcal{HOC} & & R_0 \\ \hline 0 & & 0.0 \\ 1 & & 0.9 \end{array} + \begin{array}{cc|c} \mathcal{W} & & R_1 \\ \hline 0 & & 0.1 \\ 1 & & 0.0 \end{array}$$

Arbres de décision :



Ensembles de règles de valeur :

$$R(s) = \begin{vmatrix} \mathcal{HOC} = 1 : & 0.9 \\ \mathcal{W} = 0 & : 0.1 \end{vmatrix} = |\mathcal{HOC} = 1 : 0.9| + |\mathcal{W} = 0 : 0.1|$$

Figure 1.9. *Représentation de la fonction de récompense* R *décomposée en une somme de fonction de récompense dont la portée est restreinte à une seule variable du problème.*

Comme nous l'avons décrit dans le paragraphe 1.2.3.4, la fonction de récompense du problème *Coffee Robot* peut être décomposée en une somme de deux fonctions

dont la portée n'est restreinte qu'à une seule variable du problème. Plusieurs représentations peuvent être utilisées pour représenter les fonctions composant la fonction de récompense, notamment une forme tabulaire, d'arbres de décision ou d'un ensemble de règles de valeur.

La figure 1.9 montre que deux configurations sont possibles, soit en regroupant les règles au sein d'un même ensemble pour ne définir qu'une seule fonction, soit en séparant les règles dans deux ensembles différents pour définir deux fonctions différentes. Enfin, on remarque que, même sur cet exemple simple, les règles de valeur permettent d'exploiter les indépendances contextuelles pour décrire la fonction de récompense du problème *Coffee Robot*, contrairement aux arbres de décision. En effet, les arbres de décision requièrent la représentation des feuilles contenant la valeur 0, ce qui n'est pas le cas des règles.

1.3.3.4. *Représentation d'une politique*

Pour représenter une politique π de façon compacte, [GUE 03b] reprennent une technique présentée par [KOL 00]. Plutôt que d'utiliser un arbre de décision $\mathsf{Tree}\,[\pi]$ ou un ADD $\mathsf{ADD}\,[\pi]$ pour représenter une définition structurée de π, une action par défaut est choisie *a priori* dans le FMDP et la politique est représentée comme une liste de décision ordonnée. Chaque élément de la liste est composé de trois informations différentes : un contexte indiquant si la décision peut être prise étant donné un état s, l'action à exécuter si la décision est prise et enfin le bonus indiquant la récompense espérée supplémentaire pour cette décision comparée à la récompense espérée de l'action par défaut. Le dernier élément de la liste est toujours l'action par défaut, associée à un contexte vide (pour que ce dernier élément représente la décision par défaut à prendre si aucun autre n'est cohérent avec l'état) et un bonus de 0. Une politique π représentée sous la forme d'une liste de décision est notée $\mathsf{List}\,[\pi]$. Le tableau 1.1 montre l'exemple d'une politique dans le problème *Coffee Robot* dont l'action par défaut est $\mathcal{G}_{\text{O}}$.

On peut remarquer que la politique représentée tableau 1.1 n'est pas simplifiée. En effet, par exemple, la règle 3 peut être agrégée avec la règle 1 puisque ces deux règles ont le même contexte (la règle 3 ne sera jamais utilisée puisque la règle 1 sera nécessairement utilisée avant). De plus, contrairement aux arbres de décision ou aux ADD, le nombre de tests réalisés pour déterminer l'action à exécuter peut être supérieur au nombre de variables décrivant le problème.

Enfin, pour certains problèmes de grande taille, quelle que soit la méthode de planification utilisée, une représentation explicite de la politique optimale, même factorisée, est impossible puisqu'il est nécessaire pour chaque état d'évaluer toutes les variables du problème afin de déterminer la meilleure action à réaliser par l'agent. C'est la raison pour laquelle [GUE 03b] propose des algorithmes ne nécessitant pas une représentation explicite de la politique du problème.

	Contexte	Action	Bonus
0	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 0 \wedge \mathcal{O} = 1$	$\mathcal{D}$elC	2.28
1	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 0 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 0 \wedge \mathcal{O} = 0$	$\mathcal{B}$uyC	1.87
2	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 1 \wedge \mathcal{O} = 1$	$\mathcal{D}$elC	1.60
3	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 1 \wedge \mathcal{W} = 1 \wedge \mathcal{O} = 1$	$\mathcal{D}$elC	1.45
4	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 0 \wedge \mathcal{O} = 1$	$\mathcal{D}$elC	1.44
5	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 0 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 1 \wedge \mathcal{O} = 0$	$\mathcal{B}$uyC	1.27
6	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 0 \wedge \mathcal{W} = 1 \wedge \mathcal{O} = 0$	$\mathcal{B}$uyC	1.18
7	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 0 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 0 \wedge \mathcal{O} = 0$	$\mathcal{B}$uyC	1.18
8	$\mathcal{HOC} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 0$	$\mathcal{D}$elC	0.84
9	$\mathcal{HOC} = 0 \wedge \mathcal{HRC} = 0 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 0 \wedge \mathcal{O} = 1$	$\mathcal{G}$etU	0.18
10	$\mathcal{HOC} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 1$	$\mathcal{D}$elC	0.09
11	$\emptyset$	$\mathcal{G}$o	0.00

Tableau 1.1. *Représentation d'une politique $\pi(s)$ sous la forme d'une liste de décision* $\mathsf{List}\,[\pi]$ *(avec $\mathcal{G}$o l'action par défaut)*

1.3.3.5. *Représentation de la fonction de valeur*

Nous avons vu qu'un MDP pouvait s'écrire sous la forme d'un programme linéaire de la façon suivante (paragraphe **??** du volume 1, LP 1) :

$$\begin{array}{ll} \text{Pour les variables :} & V(s), \forall s \in S\ ; \\ \text{Minimiser :} & \sum_s \alpha(s) V(s)\ ; \\ \text{Avec les contraintes :} & V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \\ & \forall s \in S, \forall a \in A. \end{array} \qquad \text{(LP 2)}$$

Cependant, une telle représentation pose un problème de complexité, aussi bien en le nombre de variables à déterminer, qu'en le nombre de termes de la somme de la fonction objectif, ou en le nombre de contraintes.

Une solution pour éviter l'explosion combinatoire concernant le nombre de variables à déterminer et le nombre de termes dans la fonction à minimiser est l'approximation de la fonction de valeur par une *combinaison linéaire* proposée par [BEL 63] (voir le chapitre **??**, volume 2). L'espace des fonctions de valeur approchées $\tilde{V} \in \mathcal{H} \subseteq \mathbb{R}^n$ est défini *via* un ensemble de *fonctions de base*, ou *basis functions*, dont la portée est limitée à un petit nombre de variables :

DÉFINITION 1.8.– *Fonction de valeur linéaire*
Une fonction de valeur linéaire $\tilde{V}$ sur un ensemble $H = \{h_0, \ldots, h_k\}$ de fonctions de base est une fonction telle que $\tilde{V}(s) = \sum_{j=1}^{k} w_j h_j(s)$ avec $w \in \mathbb{R}^k$.

Cette approximation peut être utilisée pour redéfinir le programme linéaire simplement en remplaçant la fonction de valeur à déterminer par son approximation [SCH 85] :

$$
\begin{array}{ll}
\text{Pour les variables :} & w_1, \ldots, w_k \text{ ;} \\
\text{Minimiser :} & \sum_s \alpha(s) \sum_{i=1}^{k} w_i h_i(s) \text{ ;} \\
\text{Avec les contraintes :} & \sum_{i=1}^{k} w_i h_i(s) \geq R(s,a) + \\
 & \qquad \gamma \sum_{s'} P(s'|s,a) \sum_{i=1}^{k} w_i h_i(s') \\
 & \forall s \in S, \forall a \in A.
\end{array}
\qquad \text{(LP 3)}
$$

Ainsi, plutôt que de déterminer la fonction de valeur dans l'espace complet des fonctions de valeur, l'espace de recherche est réduit à l'espace des valeurs pour l'ensemble des coefficients utilisés dans la combinaison linéaire. De plus, le fait de limiter la portée des fonctions de base permet d'exploiter les indépendances relatives aux fonctions de base.

On peut donc remarquer que le nombre de variables à déterminer du programme linéaire n'est plus le nombre d'états possibles mais le nombre de coefficients dans l'approximation linéaire. Cependant, le nombre de termes dans la fonction à minimiser et le nombre de contraintes sont toujours égaux aux nombres d'états dans le problème.

Pour un tel programme, une solution existe si une fonction de base constante est incluse dans l'ensemble des fonctions de base [SCH 85]. Nous supposons donc qu'une telle fonction h_0, telle que $h_0(s) = 1, \forall s \in S$, est systématiquement incluse à l'ensemble des fonctions de base. De plus, il est important de noter que le choix des pondérations d'intérêt $\alpha(s)$ influe sur la qualité de l'approximation [FAR 01].

En plus de la diminution de la complexité du programme linéaire, une telle approximation de la fonction de valeur permet d'exploiter à la fois les indépendances fonctionnelles et certaines régularités de la structure de la fonction de valeur. Dans le problème *Coffee Robot*, la figure 1.10 montre un exemple de décomposition additive de la fonction de valeur approchée permettant d'exploiter une régularité que des représentations telles que les arbres de décision et les ADD ne pouvaient pas utiliser.

La définition d'une fonction de valeur $\mathsf{Tree}[V]$ du problème est décomposée en deux fonctions de base $\mathsf{Tree}[h_1]$ et $\mathsf{Tree}[h_2]$[8] et permet une approximation de $\mathsf{Tree}[V_\pi]$ dont l'erreur est inférieure à 1. La propriété de décomposition additive est exploitée puisque, plutôt que de contenir 18 feuilles, cette représentation ne nécessite que 11 feuilles pour les deux arbres (soit 20 nœuds au total, au lieu de 35 nœuds pour $\mathsf{Tree}[V_\pi]$). Cette décomposition contient deux fonctions de base (trois en comptant la fonction constante h_0), donc trois coefficients, w_0, w_1 et w_2, sont à déterminer dans le programme linéaire 2.

8. Ces deux fonctions $\mathsf{Tree}[h_1]$ et $\mathsf{Tree}[h_2]$ ont été obtenues à partir de l'arbre de décision représentant la fonction de valeur $\mathsf{Tree}[V_\pi]$ dans le problème *Coffee Robot*, figure 1.6.

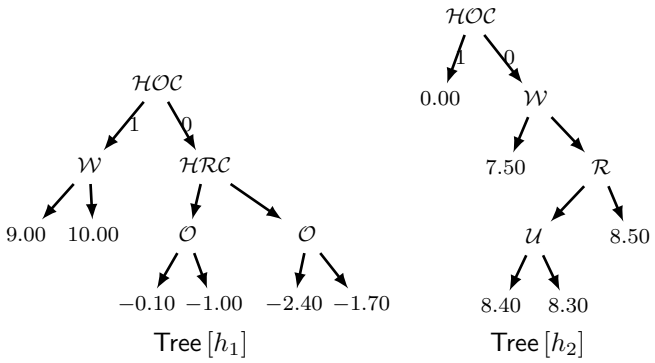


Figure 1.10. *Exemple de décomposition de la fonction de valeur du problème* Coffee Robot *sous la forme de deux arbres de décision représentant deux fonctions de base permettant de calculer la politique $\pi(s)$ (tableau 1.1). La fonction de valeur optimale approchée est : $\tilde{V}^*(s) = 0.63 \cdot \mathsf{Tree}\,[h_0] + 0.94 \cdot \mathsf{Tree}\,[h_1] + 0.96 \cdot \mathsf{Tree}\,[h_2]$. L'arbre $\mathsf{Tree}\,[h_0]$ n'est pas illustré puisqu'il définit une fonction constante et ne contient donc qu'une seule feuille égale à 1.*

Enfin, lorsque la fonction de récompense possède une décomposition additive, comme c'est le cas dans le problème *Coffee Robot*, il semble naturel que la fonction de valeur du problème possède également cette propriété. Cependant, ces deux propriétés ne sont pas nécessairement corrélées. En effet, bien qu'une fonction de récompense puisse ne présenter aucune décomposition additive, une combinaison linéaire de fonctions de base peut quand même permettre de déterminer avec une faible erreur d'approximation les fonctions de valeur du problème. Une telle représentation est donc plus générale que les représentations sous forme d'arbre de décision ou d'ADD proposées par SPI ou SPUDD [GUE 03b]. Réciproquement, des représentations compactes des fonctions de transition et de récompense n'impliquent pas non plus une représentation compacte de la fonction de valeur [KOL 99, MUN 00, LIB 02].

1.3.3.6. *Algorithmes*

Les algorithmes [GUE 03b] permettent ainsi, à partir de la définition d'un problème sous la forme d'un FMDP, de générer le programme linéaire associé afin de calculer une approximation de la fonction de valeur du problème. De plus, des algorithmes sont aussi proposés afin d'obtenir une représentation de la politique (sous la forme exposée dans la section 1.3.3.4). Cependant, une telle représentation peut s'avérer trop coûteuse, c'est la raison pour laquelle les auteurs proposent une autre alternative : une fois la fonction de valeur calculée, pour un état donné, il est facile de calculer la valeur d'action pour chaque action, permettant ainsi de trouver la meilleure action à exécuter pour cet état. Ainsi, une représentation explicite de la politique est évitée. Le lecteur pourra consulter [GUE 03b] pour obtenir une description exhaustive de ces algorithmes.

1.4. Conclusion et perspectives

La résolution efficace de problèmes décisionnels de Markov de grande taille dans un cadre factorisé est un domaine de recherche très actif. Plusieurs extensions de ce cadre ont été proposées, notamment dans le cadre partiellement observable [POU 05] et dans celui de l'apprentissage par renforcement. Cette seconde extension se justifie par le fait que définir complètement la fonction de transition et de récompense d'un FMDP peut s'avérer fastidieux, voir impossible dans certains cas.

Dans ce cadre, la première famille d'algorithmes est l'adaptation directe des algorithmes d'apprentissage par renforcement présentés dans le paragraphe **??**, volume 1, nommément DBN-E^3 [KEA 99], *factored* R-MAX [STR 07] et *factored* I.E. [STR 07]. Ces algorithmes supposent que la structure des DBN du FMDP est connue, mais pas quantifiée. Les algorithmes proposent alors un apprentissage permettant d'obtenir une politique proche d'une politique optimale du FMDP en un temps fini.

La deuxième famille d'algorithmes ne fait aucune hypothèse sur la structure de la fonction de transition et de récompense du FMDP. En s'inspirant de l'apprentissage par renforcement indirect (voir volume 1, section **??**), de récents travaux proposés par [DEG 07] proposent d'apprendre la structure des problèmes à partir de l'expérience d'un agent et en utilisant l'induction d'arbres de décisions. Bien que les résultats expérimentaux soient intéressants, une preuve mathématique de cette approche n'existe pas encore. Nous renvoyons le lecteur à [DEG 07] pour un exposé plus précis de ces dernières méthodes.

Des recherches destinées à combler le fossé entre les approches disposant d'une preuve de convergence et celles qui en sont dépourvues sont elles aussi très actives. L'approche suivie par l'équipe de Littman dans ce cadre consiste à proposer des preuves de convergence pour des algorithmes présupposant de moins en moins de connaissances *a priori* sur la structure du FMDP qu'il s'agit de résoudre. On consultera en particulier [STR 07] sur ce point.

Enfin, une dernière extension du cadre des FMDP consiste à combiner la factorisation et une approche hiérarchique. Les travaux de thèse de Teichteil [TEI 05] se sont intéressés à ce cadre. On peut aussi citer l'algorithme VISA [JON 06] qui présente des performances comparables à celles des algorithmes de Guestrin avec des méthodes de programmation dynamique.

1.5. Bibliographie

[BAH 93] BAHAR R., FROHM E., GAONA C., HACHTEL G., MACII E., PARDO A., SOMENZI F., « Algebraic Decision Diagrams and their Applications », *Proceedings of the IEEE/ACM International Conference on CAD*, p. 188–191, 1993.

[BEL 63] BELLMAN R., KALABA R., KOTKIN B., « Polynomial Approximation - a New Computational Technique in Dynamic Programming », *Math. Comp.*, vol. 17, n°8, p. 155–161, 1963.

[BOU 95] BOUTILIER C., DEARDEN R., GOLDSZMIDT M., « Exploiting Structure in Policy Construction », *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, p. 1104–1111, 1995.

[BOU 96] BOUTILIER C., GOLDSZMIDT M., « The Frame Problem and Bayesian Network Action Representations », *Proceedings of the 11th Biennial Canadian Conference on Artificial Intelligence (AI '96)*, p. 69–83, 1996.

[BOU 99] BOUTILIER C., DEAN T., HANKS S., « Decision-Theoretic Planning : Structural Assumptions and Computational Leverage », *Journal of Artificial Intelligence Research*, vol. 11, p. 1–94, 1999.

[BOU 00] BOUTILIER C., DEARDEN R., GOLDSZMIDT M., « Stochastic Dynamic Programming with Factored Representations », *Artificial Intelligence*, vol. 121, n°1, p. 49–107, 2000.

[BRY 86] BRYANT R. E., « Graph-Based Algorithms for Boolean Function Manipulation », *IEEE Transactions on Computers*, vol. C-35, n°8, p. 677–691, 1986.

[DEA 89] DEAN T., KANAZAWA K., « A Model for Reasoning about Persistence and Causation », *Computational Intelligence*, vol. 5, p. 142–150, 1989.

[DEG 07] DEGRIS T., Apprentissage par renforcement dans les processus de décision markoviens factorisés, thèse de doctorat, université Pierre et Marie Curie - Paris 6, 2007.

[FAR 01] DE FARIAS D., VAN ROY B., « The Linear Programming Approach to Approximate Dynamic Programming », *Operations Research*, vol. 51, n°6, p. 850–856, 2001.

[GUE 01] GUESTRIN C., KOLLER D., PARR R., « Max-norm Projections for Factored MDPs », *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, p. 673–680, 2001.

[GUE 03a] GUESTRIN C., Planning Under Uncertainty in Complex Structured Environments, thèse de doctorat, Computer Science Department, Stanford University, Etats-Unis, 2003.

[GUE 03b] GUESTRIN C., KOLLER D., PARR R., VENKATARAMAN S., « Efficient Solution Algorithms for Factored MDPs », *Journal of Artificial Intelligence Research*, vol. 19, p. 399–468, 2003.

[HOE 99] HOEY J., ST-AUBIN R., HU A., BOUTILIER C., « SPUDD : Stochastic Planning using Decision Diagrams », *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, Morgan Kaufmann, San Mateo, CA, p. 279–288, 1999.

[HOE 00] HOEY J., ST-AUBIN R., HU A., BOUTILIER C., Optimal and Approximate Stochastic Planning using Decision Diagrams, Rapport n°TR-00-05, University of British Columbia, 2000.

[JON 06] JONSSON A., BARTO A., « Causal Graph Based Decomposition of Factored MDPs », *Journal of Machine Learning Research*, vol. 7, p. 2259–2301, 2006.

[KEA 99] KEARNS M., KOLLER D., « Efficient Reinforcement Learning in Factored MDPs », *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 1999.

[KOL 99] KOLLER D., PARR R., « Computing Factored Value Functions for Policies in Structured MDPs », *Proceedings 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, p. 1332–1339, 1999.

[KOL 00] KOLLER D., PARR R., « Policy Iteration for Factored MDPs », *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, p. 326–334, 2000.

[LIB 02] LIBERATORE P., « The size of MDP factored policies », *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, p. 267–272, 2002.

[MAN 60] MANNE A. S., *Linear Programming and Sequential Decisions*, Cowles Foundation for Research in Economics at Yale University, 1960.

[MUN 00] MUNDHENK M., GOLDSMITH J., LUSENA C., ALLENDER E., « Complexity of Finite-Horizon Markov Decision Process Problems », *Journal of the ACM (JACM)*, vol. 47, n°4, p. 681–720, ACM Press New York, NY, Etats-Unis, 2000.

[PEA 88] PEARL J., *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA, 1988.

[POO 97] POOLE D., « The Independent Choice Logic for Modelling Multiple Agents under Uncertainty », *Artificial Intelligence*, vol. 94, n°1-2, p. 7–56, 1997.

[POU 05] POUPART P., Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes, thèse de doctorat, University of Toronto, 2005.

[QUI 93] QUINLAN J. R., *C4.5 : Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.

[RIV 87] RIVEST R. L., « Learning Decision Lists », *Machine Learning*, vol. 2, p. 229–246, 1987.

[SCH 85] SCHWEITZER P., SEIDMANN A., « Generalized Polynomial Approximations in Markovian Decision Processes », *Journal of Mathematical Analysis and Applications*, vol. 110, p. 568–582, 1985.

[STA 01] ST-AUBIN R., HOEY J., BOUTILIER C., « APRICODD : Approximate Policy Construction Using Decision Diagrams », *Advances in Neural Information Processing Systems 13 (NIPS'00)*, p. 1089–1095, 2001.

[STR 07] STREHL A., DIUK C., LITTMAN M. L., « Efficient Structure Learning in Factored-state MDPs », *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07)*, 2007.

[TEI 05] TEICHTEIL-KONIGSBUSCH F., Stratégie d'exploration pour un aéronef autonome, thèse de doctorat, Ecole Nationale Supérieure d'Aéronautique et de l'Espace, 2005.

[ZHA 99] ZHANG T., POOLE D., « On the Role of Context-specific Independence in Probabilistic Reasoning », *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, p. 1288–1293, 1999.