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

Jean-Christophe Magnan and Pierre-Henri Wuillemin

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

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

1 Introduction

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

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

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

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

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

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

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

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

2 Plannification et Apprentissage avec des modèles factorisés

2.1 Processus Décisionnel de Markov Factorisé

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

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

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

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

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

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

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

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

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

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

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

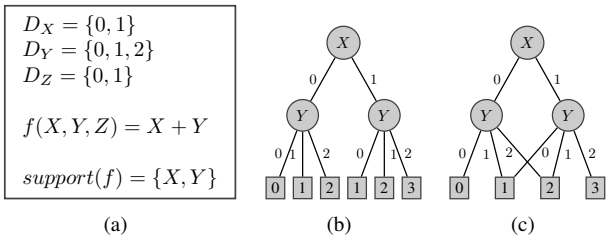


FIGURE 1 – Deux graphes de fonctions représentant la même fonction $f(X, Y, Z) = X + Y$

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

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

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

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

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

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

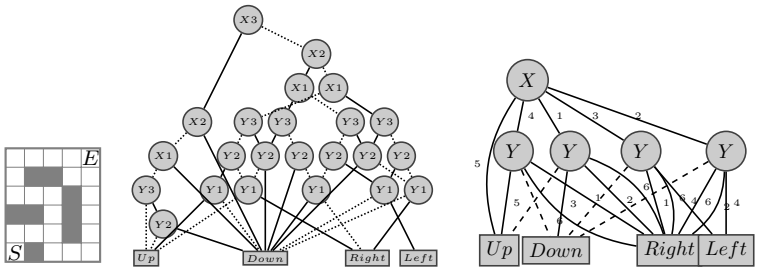
Définition 3 (Graphe de fonction ordonné)
Un graphe de fonction G_f est ordonné $\Leftrightarrow$ $\exists \succ_{G_f}$ ordre total sur le $support(f)$, t.q. $\forall$ nœuds n_1, n_2 non terminaux de G_f,

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

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

2.2 Planification et apprentissage conjoints

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



(a) (b) (c)

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

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

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

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

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

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

2.3 ITI : apprentissage incrémental d'arbre

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

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

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

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

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

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

2.4 SPITI : Planification et apprentissage avec des DTs

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

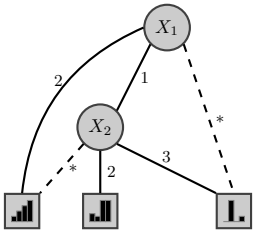


FIGURE 3 – Un MDD représentant une distribution de probabilité $P(Y|X_1, X_2)$. Les feuilles contiennent les distributions de probabilité Y. Ce MDD indique que $P(Y|X_1 = 2) = P(Y|X_1 = 1, X_2 \notin \{2, 3\})$ et que $P(Y|X_1 \notin \{1, 2\}) = P(Y|X_1 = 1, X_2 = 3)$. Ces égalités représentent des indépendances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

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

2.5 Architecture SDYNA avec MDDs

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

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

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

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

3 Apprentissage en-ligne de Diagrammes de Décision Multimodale

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

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

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

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

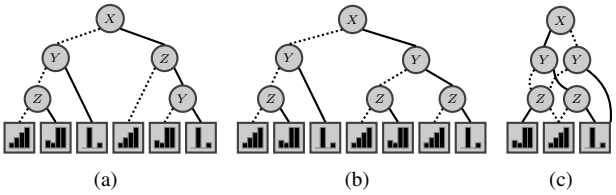


FIGURE 4 – Une distribution de probabilité représentée à l'aide d'un DT, (b) un ODT ($X \succ Y \succ Z$) et (c) un MDD (utilisant le même ordre global).

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

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

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

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

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

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

3.1 IOTI : Induction Incrémentale d'Arbre de Décision Ordonné

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

3.1.1 Sélection de Variable

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

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

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

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

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

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

3.1.2 Ajouter une nouvelle observation ξ

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

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

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

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

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

3.1.3 Mettre à jour l'ODT

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

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

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

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

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

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

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

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

3.2 De IOTI à IMDDI : génération des diagrammes de décision

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

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

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

```
1  begin
2      B = { racine R de T};                                   // frontière
3      F = X;          // Variables non insérées dans la mise à jour de l'ordre global
4      repeat
5          foreach variable X_i ∈ F do
6              p_G(X_i) = Σ_{N∈B} (|Ω_N| / |Ω|) · p_G^N(X_i);
7          end
8          V ← arg min_{X_i∈F} p_G(X_i);
9          B' ← B;
10         foreach N ∈ B do
11             if p_G^N(V) ≥ τ^1 then
12                 Installer V au nœud N;
13                 B' ← B' \ {N} ∪ ⋃_{v∈Dom(V)} c_N(v)
14             end
15         end
16         B ← B';
17         F ← F \ {V};
18     until F = ∅ or ∄X_i ∈ F et N ∈ B t.q. X_i peut être installée en N;
19 end
```

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

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

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

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

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

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

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

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

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

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

Data: un ODT T

```
1  begin
2      repeat
3          (U*, V*) = arg min_(U,V) feuilles (max(p_U^G, p_V^G));
4          if max(p_G^U*, p_G^V*) ≤ τ² then
5              Fusionner U* et V*;
6          end
7      until ∄ deux feuilles qui peuvent être fusionnées;
8      foreach X_i ∈ X, en remontant dans l'ordre global do
9          foreach pair de nœuds internes N_Xi, N'_Xi ayant X_i comme variable installée do
10             if ∀x_i ∈ Dom(X_i), c_N_Xi(x_i) = c_N'_Xi(x_i) then
11                 N_Xi et N'_Xi sont fusionnés ;
12             end
13         end
14         foreach N_Xi ayant X_i comme variable installée do
15             if ∀x_i^k, x_i^l ∈ Dom(X_i), c_N_Xi(x_i^k) = c_N_Xi(x_i^l) then
16                 Remplacer N_Xi par des arcs allant des parents à l'unique fils.
17             end
18         end
19     end
20 end
```

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

3.3 Étude de la complexité d'IMDDI

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

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

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

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

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

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

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

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

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

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

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

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

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

3.4 SPIMDDI dans un cadre SDYNA

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

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

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

4 Experiences et Observations

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

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

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

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

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

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

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

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

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

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

4.1 Qualité des modèles appris

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

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

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

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

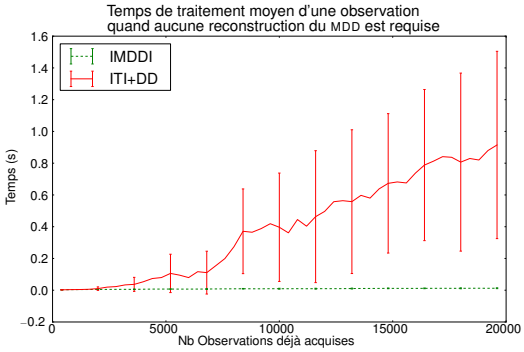


FIGURE 5 – Temps de prise en compte d'une nouvelle observation par IMDDI et ITI+DD. Avant de créer ce graphe, les itérations où l'ajout de l'observation a conduit à une modification du MDD final ont été écartées. Pour les deux algorithmes, ces temps sont mesurés lors des apprentissages des bases générées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

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

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

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

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

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

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

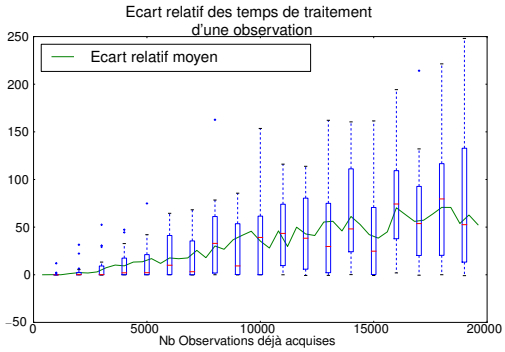


FIGURE 6 – Évolution de l'écart relatif $\frac{T_{ITI+DD}-T_{IMDDI}}{T_{IMDDI}}$ (où T_X est le temps total pour l'algorithme X pour intégrer la nouvelle observation). Pour établir ce graphe, nous nous sommes servi des modèles appris à partir des bases BNs.

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

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

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

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

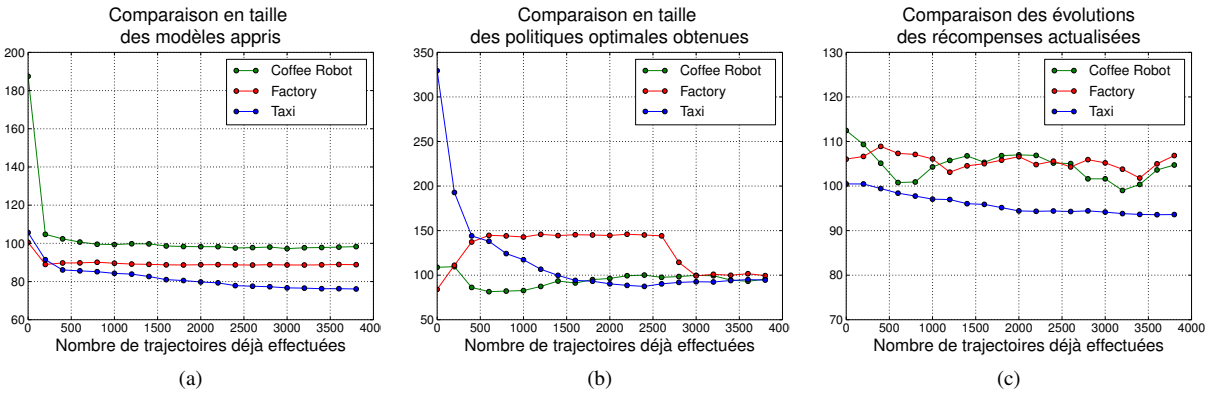


FIGURE 7 – Comparaison en % de SPIMDDI par rapport à SPITI suivant (a) la taille des modèles appris (transitions et récompenses), (b) la taille de la politique optimale plannifiée, et (c) la récompense actualisée obtenue au cours de l'apprentissage.

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

5 Conclusion

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

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

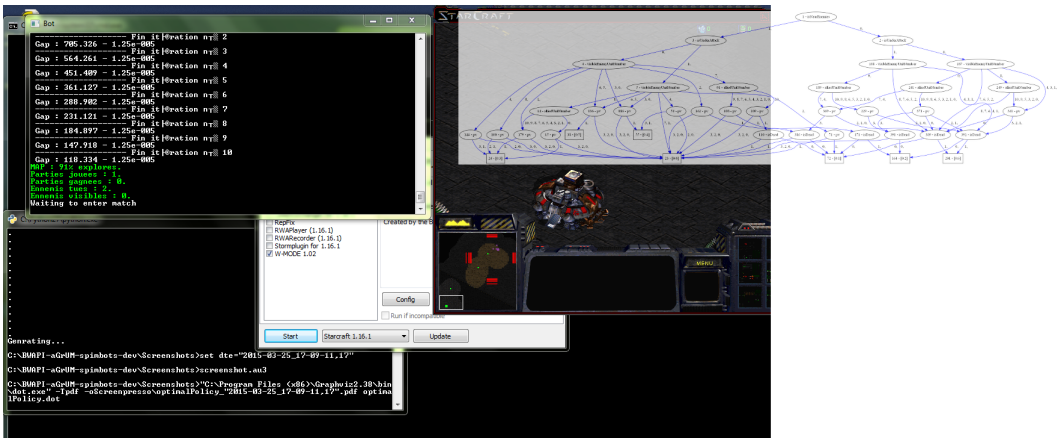


FIGURE 8 – Un aperçu de SPIMDDI utilisé dans le jeu vidéo StarCraft. En surimpression, la politique optimale obtenue après une cinquantaine de parties.

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

Références

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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