Apprentissage de modèles de comportement pour le contrôle d'exécution et la planification robotique

Guillaume Infantes

Thèse

préparée au
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

en vue de l'obtention du
Doctorat de l'Université Paul Sabatier de Toulouse
Soutenue le 5 Octobre 2006
Spécialité : Intelligence Artificielle

par
Guillaume Infantes

Apprentissage de modèles de comportements pour le contrôle d'exécution et la planification robotique

---

Soutenance le devant le jury :

Rachid Alami	**LAAS-CNRS, Toulouse**	Président du jury
Michael Beetz	**IAS Group, Münich**	Rapporteur
Louis Féraud	**IRIT, Toulouse**	Examinateur
Maria Fox	**SPG, Glasgow**	Invitée
Malik Ghallab	**LAAS-CNRS, Toulouse**	Directeur de thèse
Félix Ingrand	**LAAS-CNRS, Toulouse**	Directeur de thèse
Olivier Sigaud	**ANIMATLAB, Paris**	Rapporteur

LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex 4

Table des matières

Remerciements

Avant de commencer la partie scientifique souvent si sèche, une touche humaine et sensible s'impose. Une thèse c'est bien sûr beaucoup de travail, de nombreuses connaissances à maîtriser (enfin... à peu près), et avec de la chance quelques bonnes idées. Mais c'est également plus de trois ans d'une vie, avec tout ce que cela entend comme aventure humaine.

Alors est-il possible de résumer cela ? Certainement pas. L'usage est de remercier ceux qui ont fait partie de ces trois ans, mais comment faire cela sans résumer ? Comment penser à ces trois ans et au chemin parcouru, à ceux qui l'ont croisé, m'ont relevé lorsque j'étais tombé ou me soutiennent encore, tout cela sans nostalgie ? Comment rendre ce qui est dû, quant on doit tant à de si nombreuses personnes ? Voilà une question des plus difficiles... Alors il est possible de citer. Citer une longue liste de noms, et prétendre à l'exhaustivité. Est-il possible de citer tous ceux qui ont fait de moi et de cette thèse ce que nous sommes ? Je ne le pense pas, et que ceux qui ne seront pas explicitement nommés ne s'en formalisent pas, je m'excuse d'avance de ne pas réussir une tâche bien plus difficile qu'elle n'en a l'air. Quant aux raisons pour lesquelles je tiens à remercier, je pense que les personnes concernées les devineront.

Je tiens bien entendu à remercier mes directeurs de recherche, Malik et Félix mais aussi Félix et Malik, sans qui bien peu de choses auraient été possibles. Maria mérite également une mention spéciale de ma part. Les membres du jury de l'ultime journée ne sont pas à oublier, et en particulier Olivier avec qui je compte bien travailler, un jour...

RIA c'est aussi tout un environnement riche de talents ; Sara, Matthieu, Raja et Jackie en sont comme chacun sait des piliers porteurs. Citons également Simon, Michel, Frédéric, Florent, Patrick, et la liste devrait être encore longue. De l'autre coté de la vie d'enseignant-chercheur, citons une bien belle équipe avec Jean-Michel, Philippe, Jean-Claude, Pascal, Mario et les autres...

Et maintenant, oublions tout semblant d'ordre pour remercier Olivier, Stéphane, Thomas, Fred, Thony, Cédric, Julien, Seb, Max, Léo, Nico, Joan, Alex, Ludo, Williams, Romain, Manu, Solange, Ben et Gérard.

Plus loin de la thèse, mais au moins aussi près de moi, je tiens à tenter de montrer ma reconnaissance à Emilie, Julie, Julie et Julie, Céline, Hélène, Charles, Florent, Christophe, Philippe, Isa, Thomas, ainsi qu'à ma famille.

Finalement, comme une vie est aussi faite de coupures et de grands changements, je dois beaucoup à Hervé Raynaud et Christiane Alberti, grâce à qui je peux ne pas oublier tous ceux qui font maintenant plutôt partie de ma vie "d'avant".

Pour terminer, que ceux qui se sentent floués de ne pas apparaître dans cette page composée à la va-vite ne m'en veuillent pas trop, et qu'ils n'hésitent pas à me contacter pour que nous y remédions.

Je vous souhaite une bonne lecture.

Toulouse, le 30 novembre 2006

Chapitre I

Préliminaires

Nous avons choisi de grouper dans ce premier chapitre une partie proprement introductive à la thèse, présentant objectifs en termes généraux et applicatifs, et contributions principales ; et d'un autre coté les outils et notions dont nous aurons besoin pour notre propos. Ceux-ci se décomposent en notions sur les modèles graphiques pour le raisonnement dans l'incertain, et en outils de théorie de l'information que nous utiliserons au chapitre 3.

I.1 Introduction

Objectifs

Les visées de cette thèse sont d'explorer la création et l'utilisation de modèles intermédiaires en robotique. Nous postulons qu'entre modèles de bas niveau utilisés pour créer des comportements complexes robotiques et modèles abstraits utilisés pour la prise de décision à un niveau "intelligence artificielle", des modèles intermédiaires peuvent être d'une grande utilité.

En effet, pour créer des comportements robotiques, de nombreux composants logiciels et matériels sont profondément imbriqués. Ces composants sont construits dans un objectif de robustesse et même s'ils utilisent des modèles, ils sont eux-mêmes difficiles à modéliser par une approche de compositions de sous-modèles. À l'opposé, la prise de décision de haut niveau et les capacités d'intelligence d'une machine telles que la planification ont été étudiées depuis longtemps. Cependant, celles-ci se basent souvent sur des modèles simplifiés, les problèmes restant complexes malgré les nombreuses hypothèses simplificatrices.

Nous pensons donc qu'entre ces deux approches, des modèles intermédiaires appris peuvent être d'une grande utilité, à la fois pour obtenir des modèles plus précis pour la planification, mais aussi pour contrôler finement l'exécution même. Nous pensons que ces modèles peuvent permettre une automatisation de l'optimisation des comportements robotiques de bas niveau, jusqu'à une spécialisation permettant de dégager des comportements différents. Du point de vue de l'abstraction de l'information, le système de prise de décision générale et de haut niveau n'a pas à s'occuper de telles optimisations. Nous essaierons de proposer une prise de décision de bas niveau, visant à optimiser un comportement indépendamment d'un plan de plus haut niveau.

Cette approche nous pousse vers l'apprentissage automatique de ces modèles intermédiaires, car nous avons postulé de par nos connaissances des systèmes réels que de tels modèles ne peuvent être obtenus par des techniques classiques de modélisation. Nous pensons également que les informations utiles pour modéliser des comportements autonomes complexes doivent faire appel à des connaissances humaines, ce qui doit permettre d'obtenir des modèles intuitifs et des spécialisations comportementales interprétables.

D'autre part, les systèmes robotiques autonomes sont profondément aléatoires du fait de leur interaction avec un environnement imprévisible, mais aussi du fait de l'imprécision et du bruit inhérents à tout capteur physique (quand les algorithmes utilisés ne sont pas eux-mêmes de nature non-déterministe). De tels modèles intermédiaires doivent donc, selon nous, être intrinsèquement stochastiques. Nous pensons également qu'en les construisant nous devrons faire l'hypothèse de la présence de non-observabilités tant au niveau des données que de l'état global du système ; ceci du fait de la grande complexité des systèmes à modéliser.

Applications Visées

L'application principale que nous utilisons pour illustrer notre travail est la navigation robotique autonome robuste. Celle-ci est encore un problème important, et les méthodes et algorithmes proposés jusqu'à présent sont nombreux et aucun n'est universel. Dans [Morisset, 2002], on peut voir une façon élégante d'apprendre à utiliser différentes méthodes de navigation de façon à tirer parti de leur complémentarité. Pour notre part, nous nous intéressons de plus à la construction de modèles utilisables pour différentes applications comme l'optimisation, le suivi et le contrôle ou encore la planification.

Dans toutes nos expérimentations, nous utilisons une seule modalité de navigation, de type réactif. Nous cherchons à modéliser finement cette modalité en fonction de son environnement, et à dégager plusieurs comportements possibles à partir de cette seule modalité, en changeant ses paramètres. Nous essaierons d'obtenir différents comportements suivant des critères à définir, et tenterons de faire rentrer l'homme dans leur construction.

Cette modalité de navigation permet d'éviter des obstacles sur une base réactive, et est bien adaptée pour des environnements fortement encombrés (dynamiques ou non). C'est une modalité construite pour des environnements plans, elle utilise un modèle bi-dimensionnel et ne prend pas du tout en compte d'aspects de stabilité du robot. Elle ne peut donc être utilisée qu'en environnement intérieur structuré (appartements, bureaux) ou extérieur plat (parking etc).

Nous pensons que cette approche est applicable à d'autres comportements robotiques, même si nous n'avons pas eu le temps d'adapter notre méthodologie à d'autres exemples.

Dans un premier temps, pour une modélisation pure avec états cachés, nous nous intéresserons aux modèles de Markov cachés. Puis, dans le but d'introduire de la contrôlabilité dans le modèle, nous passerons à des représentations plus structurées type réseaux bayésiens dynamiques. Nous illustrerons que ce type de modèles est utilisable pour une grande variété d'applications.

Plan

Dans la suite de ce chapitre, nous allons passer en revue des notions de raisonnement et de décision dans l'incertain, nécessaires à notre propos.

Dans le deuxième chapitre, nous allons montrer comment construire un modèle stochastique à état discret à partir des données brutes. Nous allons passer en revue les aspects sélection des données intéressantes, leur traitement, et comment apprendre un tel type de modèle. Nous montrerons alors un algorithme original pour déterminer les états cachés en se servant à la fois de méthodes automatiques et de la connaissance humaine extérieure, puis montrerons une application à la navigation autonome.

Dans une troisième partie, nous passerons à des modèle plus expressifs, et expliquerons pourquoi les calculs exacts ne sont pas possibles à réaliser sur des modèles plus complexes. Nous montrerons une formalisation originale de l'apprentissage paramétrique de tels modèles à partir des données, et passerons en revue différentes méthodes d'apprentissage de la causalité interne au modèle. Ensuite viendrons des pistes pour un apprentissage incrémental et une application à la navigation autonome.

Le chapitre suivant s'intéressera à la prise de décision à partir de ce type de modèles complexes. Nous expliquerons pourquoi dans une application réelle les approches théoriques ne sont pas les meilleures, et montrerons une approche pragmatique originale, et ses possibilités en terme de décision de très haut niveau. Une application à l'optimisation de plusieurs critères différentes d'un comportement robotique autonome sera détaillée.

Enfin, dans le dernier chapitre, nous montrerons d'autres applications de ce type de modèles causaux, en faisant un rapide tour d'horizon des méthodes utilisées. Viendrons nos propres résultats en reconnaissance de gestes et d'états mentaux. Nous discuterons alors une nouvelle approche pour modéliser des systèmes réels.

Viendrons alors conclusions et perspectives pour ces approches.

Contributions

Nous soulignons que les travaux présentés dans cette thèse s'appuient sur toute une équipe de recherche. En effet, la possibilité d'expérimenter sur des robots réels ne peut être que très difficilement envisagée sans de solides acquis antérieurs et de nombreuses compétences et contributions durant la thèse, telles que par exemple celles décrites dans [Fleury, Herrb et Chatila, 1997; Alami, Chatila, Fleury, Ghallab et Ingrand, 1998; Alami et al., 2000]. D'autre part, les travaux sur les modèles de Markov cachés (chapitre II) ont été fait en collaboration avec et sous la direction du professeur Maria Fox en séjour sabbatique au laboratoire pendant la première année de la thèse de l'auteur.

Les principales contributions montrées dans cette thèse sont de plusieurs ordres.

Nous présentons exhaustivement comment construire différents types de modèles stochastiques pour des systèmes réels, des données brutes au modèle de haut niveau final. Ce procédé nécessite l'utilisation de nombreux outils qui sont présentés dans cette thèse.

Ce manuscrit explore la détermination et la définition des états cachés dans un système non observable, en proposant un algorithme original utilisant à la fois la connaissance humaine intuitive et les connaissances computationnelles.

Du coté de la construction de modèles à causalité explicite, nous n'avons jamais trouvé de formalisation d'algorithme pour l'apprentissage paramétrique. Cette thèse intègre une telle formalisation. Nous pensons par ailleurs que cette formalisation conduit à un algorithme légèrement différent et meilleur que ceux utilisés par ailleurs sans être formalisés.

Nous présentons également une façon simple et pragmatique d'optimiser les comportements du robots suivant plusieurs critères, et plaçons les bases d'une véritable gestion de la connaissance pour le contrôle des comportements d'un robot.

Nous présentons également des applications inédites de ce genre de modèle et des idées plus générales sur la gestion de la dynamique interne d'un système réel, qui est généralement peu prise en compte.

I.2 Raisonnement dans l'incertain

Dans cette section, nous présentons plusieurs cadres généraux largement utilisés pour modéliser les incertitudes, et pour prendre des décisions en présence de ces incertitudes.

Réseau Bayésiens

Les réseaux bayésiens ou réseaux probabilistes (BN) sont une façon intuitive et graphique de représenter la connaissance dans des domaines incertains. Ils permettent de représenter les indépendances entre événements, ainsi que leurs probabilités d'apparition.

Un BN est un graphe orienté dans lequel chaque nœud est quantifié par des informations de probabilités. Formellement un BN est composé des éléments suivants :

- Un ensemble de variables aléatoires qui forment les nœuds du réseau. Ces variables peuvent être discrètes ou continues ;
- Un ensemble d'arêtes orientées qui lient des paires de nœuds. Si un lien est orienté de X vers Y, on dit que X est parent de Y ;
- Chaque nœud X_i a une distribution de probabilités conditionnelles $P(X_i|parents(X_i))$ qui quantifie probabilités d'apparition des valeurs d'un nœud en fonction de ses parents ;
- le graphe n'a pas de cycle orienté (c'est un graphe orienté acyclique, ou directed acyclic graph ou encore DAG).

La topologie du réseau spécifie les indépendances conditionnelles du domaine. Le sens intuitif d'une flèche de X vers Y est que X a une influence directe sur Y. Lorsque l'on spécifie un réseau, on donne d'abord les influences et ensuite les probabilités conditionnelles de chaque variable connaissant ses parents.

Sur la figure I.1, exemple tiré de [Russell et Norvig, 2003a], on peut voir que l'alarme d'une maison peut se déclencher pour deux causes différentes, un orage ou une effraction. Deux voisins peuvent réagir à cette alarme et appeler la police. John confond le bruit de l'alarme avec la sonnerie de son propre téléphone, et il peut lui arriver d'appeler la police même si l'alarme ne

sonne pas. Mary, quant à elle, a l'habitude d'écouter de la musique forte et n'entend pas très souvent l'alarme.

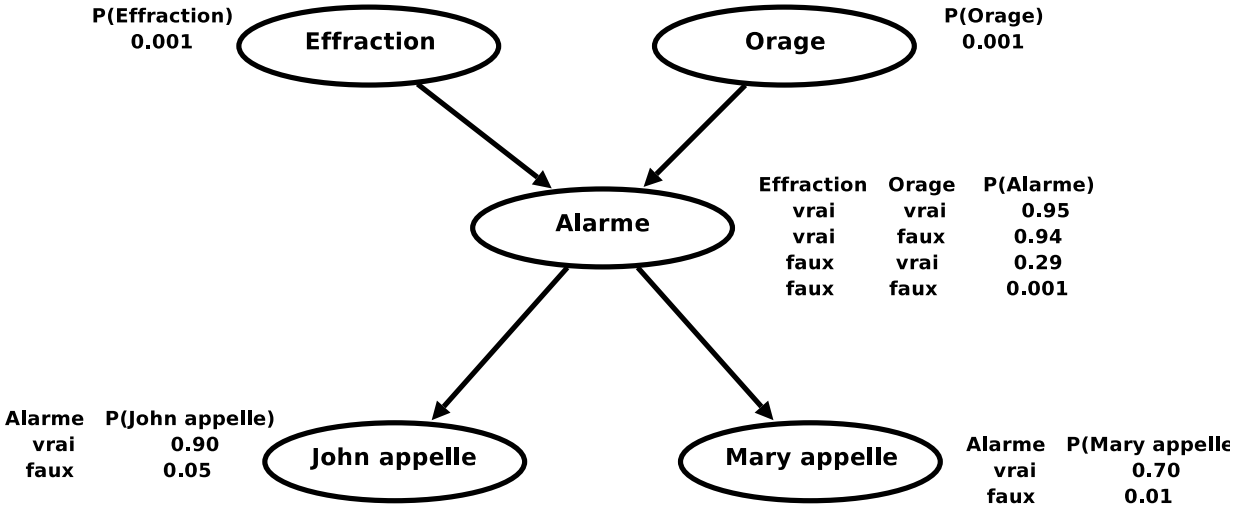


FIG. I.1: Exemple de réseau bayésien

La causalité est alors simple et intuitive. Par contre au niveau des quantifications, des remarques s'imposent. Dans le domaine de ce BN, on ne modélise pas le fait que le téléphone de John sonne ou que Mary soit en train d'écouter de la musique forte. Ces données sont capturées dans les tables de probabilités conditionnelles, ou John a un taux de faux positifs relativement élevé, et Mary un taux de faux négatifs élevé aussi. La cause de ces taux élevés n'est pas explicitée. Une fois que le BN est entièrement spécifié (y compris les connaissances a priori sur les effractions et les orages), on peut alors calculer les probabilités de toutes les variables, et même les probabilités jointes de toutes les combinaisons de variables. Le modèle est complet.

La construction d'un BN n'est pas simple. En effet, d'autres structures sont possibles pour représenter les mêmes probabilités jointes sur toutes les variables. Si l'on décide de construire un réseau pour représenter la même connaissance en introduisant des variables dans un mauvais ordre (par exemple *Mary appelle*, *John appelle*, *Alarme*, *effraction*, *orage*) on a alors le processus illustré sur la figure I.2 :

1. On ajoute *Mary appelle* ;
2. On ajoute *John appelle* : si Mary appelle, cela veut dire que l'alarme s'est sûrement déclenchée, et donc le fait que Mary appelle rend probable l'appel de John, ces deux variables ne sont donc pas indépendantes, et on introduit un lien causal ;
3. On ajoute *alarme* : clairement, cette variable est corrélées aux différents appels, et on introduit deux liens causaux ;
4. *effraction* : si l'on sait si l'alarme s'est déclenchée ou non, avoir des informations sur les appels ne nous donne pas plus d'information sur l'éventualité d'une effraction, tout au plus pourra-t-on déduire si Mary écoutait de la musique ou pas ;

5. *orage* : si l'alarme est déclenchée, il est plus probable qu'il y ait un orage ; mais si on sait qu'il y a eu une effraction et que l'alarme est déclenchée, alors la probabilité d'un orage est plus faible. *Orage* est donc dépendant de deux parents.

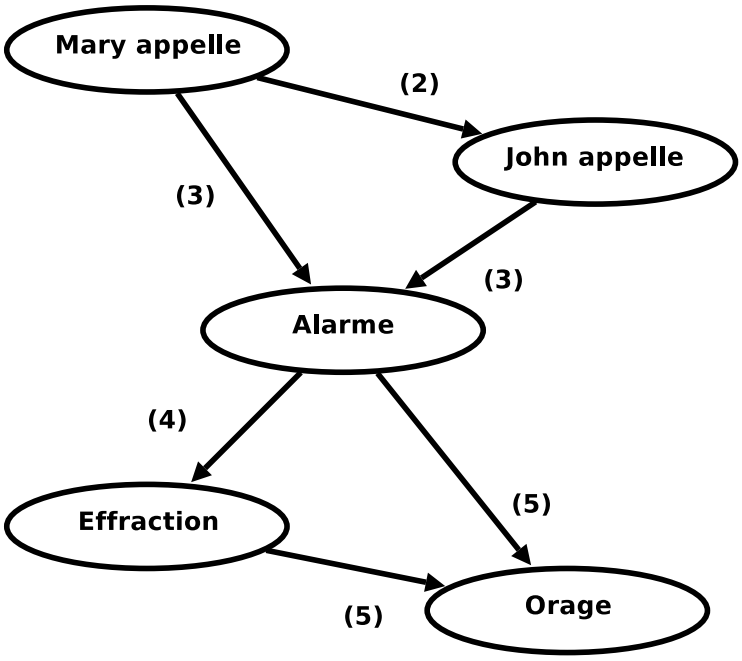


FIG. I.2: Le même domaine, représenté par un autre BN

On constate alors que le BN construit contient les mêmes informations, mais est plus complexe. Ce problème est très important pour l'apprentissage automatique de BN. Un BN minimal est un BN dans lequel les liens orientés respectent la causalité au sens intuitif du terme. Il ne faut pas perdre de vue que les liens orientés représentent uniquement des probabilités conditionnelles, qui n'ont rien à voir avec la causalité.

Au niveau des indépendances entre variables, on a deux propriétés :

– un nœud est indépendant de ses non descendants, étant donné ses parents. Dans la figure I.1, *John appelle* est indépendant de *effraction* et *orage*, étant donné la valeur de *alarme* ;
– un nœud est conditionnellement indépendant de tous les autres nœud du réseau étant donnés ses parents, ses enfants, et les autres parents de ses enfants. *effraction* est indépendant de *John appelle* et *Mary appelle*, étant donnés *alarme* et *orage*.

Processus Décisionnels de Markov

La prise de décision dans des domaines stochastiques est un problème étudié depuis longtemps. Si on peut savoir à chaque instant l'état de l'agent, et que seules ses actions sont stochastiques, on parle de processus décisionnel de Markov (MDP pour Markov Decision Process). Dans un tel modèle, l'agent peut choisir une action parmi plusieurs, mais les effets de ces ac-

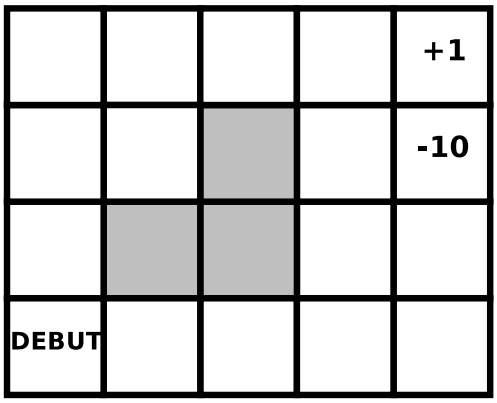
tions peuvent amener suivant un modèle de transition probabiliste à plusieurs états différents, en fonction de l'état de départ.

Des récompenses sont associées à certains états, et trouver une solution au MDP consiste donc à trouver une politique optimale qui associe à chaque état l'action à effectuer afin de maximiser la somme des récompenses sur un horizon fini. Ce problème se généralise sur un horizon infini en ne cherchant pas à maximiser la somme des récompenses (qui n'a a priori pas de borne supérieure), mais la somme des récompenses pondérées par un facteur d'amortissement tendant vers 0 avec le temps. Ainsi on préfère les récompenses à court terme et néglige les récompenses à l'infini. (Il est intéressant de noter que choisir une valeur de ce facteur d'amortissement est complètement non-intuitif suivant les domaines étudiés).

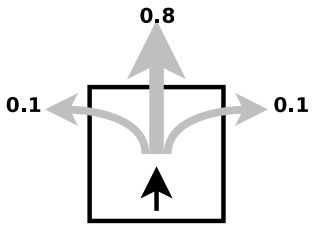
Formellement, un processus décisionnel de Markov se compose de :

- un ensemble d'états possibles S de l'agent ;
- un ensemble d'actions possibles A pour l'agent ;
- un modèle de transition de $T(s, a, s')$, avec $s, s' \in S$ et $a \in A$, qui donne la probabilité pour l'agent d'arriver en s' partant de s et effectuant l'action a ;
- une fonction de récompense $R(s)$ qui associe à chaque état une valeur coût ou récompense, significative de la mission ou du critère propre au problème modélisé.

L'exemple le plus courant de MDP consiste en un robot se déplaçant dans un monde "grille" glissant (figuré sur la figure I.3(a)). Ses actions sont stochastiques (figure I.3(b)) telles que si le robot veut se déplacer dans une direction, il a une probabilité de "glisser" vers une case adjacente. S'il essaie d'avancer dans un obstacle (ou un bord du monde), il reste où il est. Il peut néanmoins avant chaque décision observer sur quelle case il est. Cet exemple présente un état but (une récompense), et une case dangereuse pour le robot, qu'une politique solution doit faire éviter au robot.



(a) Monde grille de déplacement du robot



(b) Modèle de transition (hors obstacle)

FIG. I.3: Exemple typique de MDP

Si on donne à ce problème un état de départ de l'agent, et qu'il n'y a qu'un seul état but, on est alors dans le cadre de la planification contingente. Dans ce cadre, une solution n'est pas une politique, mais une suite d'actions propre à maximiser la probabilité d'atteindre le but. D'autre

part, le problème se pose en cas d'objectifs multiples et de hiérarchie entre eux, car ce type de modèles est entièrement statique et ne permet pas de spécifier si un objectif a été atteint, de manière à passer au suivant ou simplement d'un quelconque changement dans la fonction récompense. Tous ces problèmes sont expliqués dans [Teichteil-Königsbuch, 2005].

Le cadre le plus courant est présenté ici, c'est celui des MDP d'ordre 1, c'est-à-dire que le modèle de transition T s'exprime en $T(s, a, s')$. Cela signifie que l'effet d'une action donnée dépend uniquement de l'état courant, et pas d'autres états passés, ou encore que toute l'information est contenue dans l'état courant. Ce modèle se généralise à des ordres supérieurs, où un modèle de transition peut s'exprimer en fonction d'une partie de l'historique. Ces modèles sont peu étudiés car ils sont exprimables en MDP d'ordre 1, pour peu que l'on multiplie le nombre d'états possibles. Par exemple, si l'on a deux états possibles s_1 et s_2 pour un agent, mais que le modèle de transition est d'ordre 2, c'est-à-dire dépend de l'état courant et du précédent, il suffit de s'intéresser au problème dans lequel l'état courant est exprimé par $s_\alpha = (s_1; s_1), s_\beta = (s_1; s_2), s_\gamma = (s_2; s_1), s_\delta = (s_2; s_1)$ etc... Si l'agent est en s_α, cela signifie que son état courant et son état à l'instant d'avant sont tous les deux s_1, s'il est en s_β que son état courant est s_2 et son état passé s_1 et ainsi de suite.

Réseaux décisionnels

Si les réseaux bayésiens permettent de représenter l'incertitude de façon simple, la notion de contrôlabilité en est complètement absente. Les diagrammes d'influence ou réseaux décisionnels [Howard et Mathson, 1984] ajoutent aux BN des variables contrôlables et des utilités.

Ils alors composés de trois types de nœuds :

- Les variables aléatoires (ovales), qui se comportent exactement comme les variables des réseaux bayésiens ;
- les nœuds décisionnels (rectangles), qui sont des variables dont un agent peut fixer la valeur ;
- les nœuds d'utilité (losanges), qui représentent coûts et récompenses associées à des valeurs des variables parents.

Les liens orientés gardent la même signification, à savoir représentent des probabilités conditionnelles.

Typiquement, on peut représenter une étape d'un processus décisionnel de Markov sous la forme du DN de la figure I.4.

Résoudre un MDP ne consiste pas seulement à maximiser l'utilité à l'instant suivant, mais sur un horizon h potentiellement infini. Pour représenter cela, il faudrait un DN de longueur h. Mais comme dans un MDP les probabilités quantifiant les liens causaux sont stationnaires, on peut représenter le MDP sous forme "dynamique" comme sur la figure I.5. Un réseau décisionnel dynamique est une façon de replier la représentation d'un DN de taille infinie, pour peu que les probabilités de transition soient stationnaires. Nous reviendrons aux sections III.1 et V.1 sur une définition formelle de cette représentation.

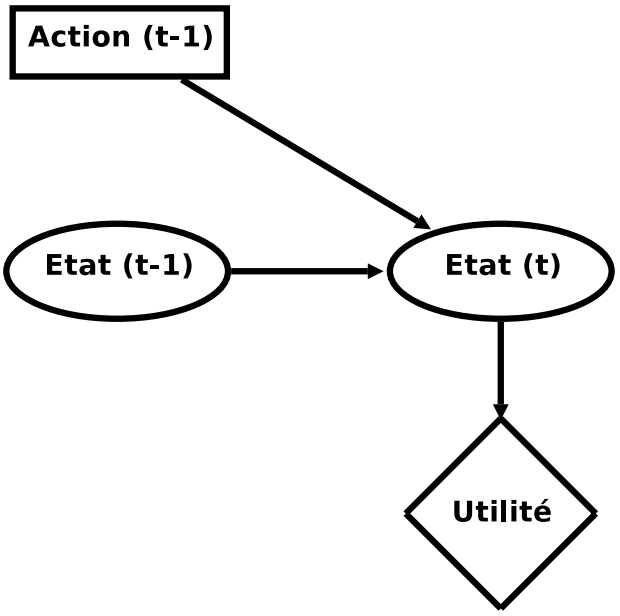


FIG. I.4: Représentation d'une étape de MDP sous forme de réseau décisionnel

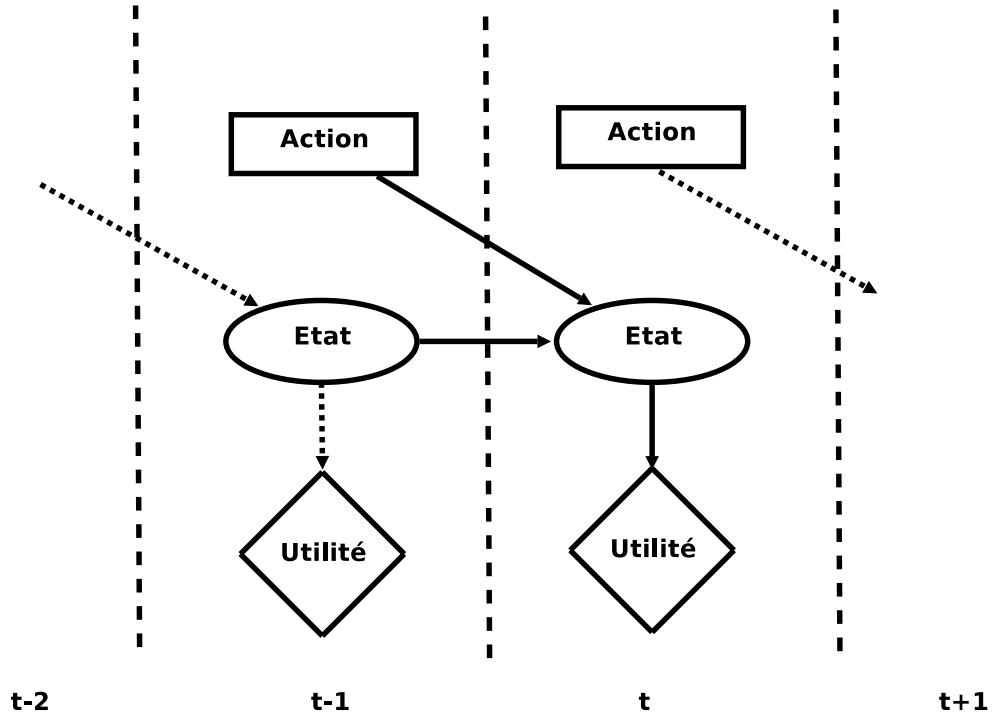


FIG. I.5: Représentation d'un MDP sous forme de réseau décisionnel dynamique

Tout n'est pas observable...

Jusqu'à présent, nous n'avons pas considéré l'apprentissage de ce type de modèles. On imagine aisément que si l'on a la structure d'un BN, et suffisamment de données, on peut déduire

statistiquement les probabilités quantifiant les liens causaux.

Cependant, ceci n'est vrai que si les données sont complètes, c'est-à-dire qu'il n'y a pas de variable cachée à l'observateur qui veut construire un modèle. Or pour la plupart des systèmes réels, c'est le cas. Si l'on considère l'exemple de l'alarme, et qu'on ne peut pas savoir si l'alarme a sonné ou non, on peut imaginer apprendre le BN de la figure I.6. Dans certains cas (comme illustré sur la figure I.7), la complexité peut être beaucoup plus grande.

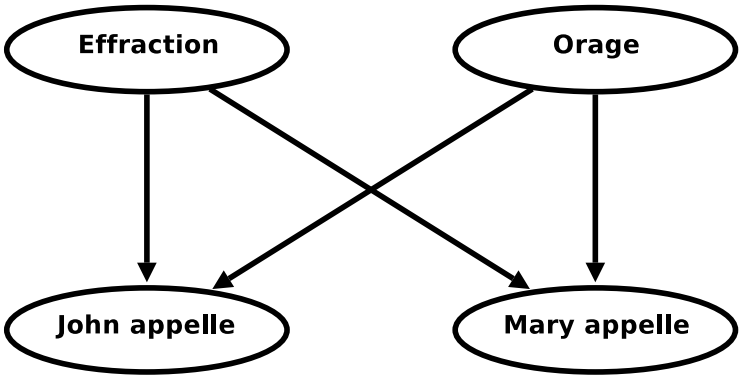


FIG. I.6: Réseau bayésien de l'alarme sans tenir compte de la variable alarme cachée

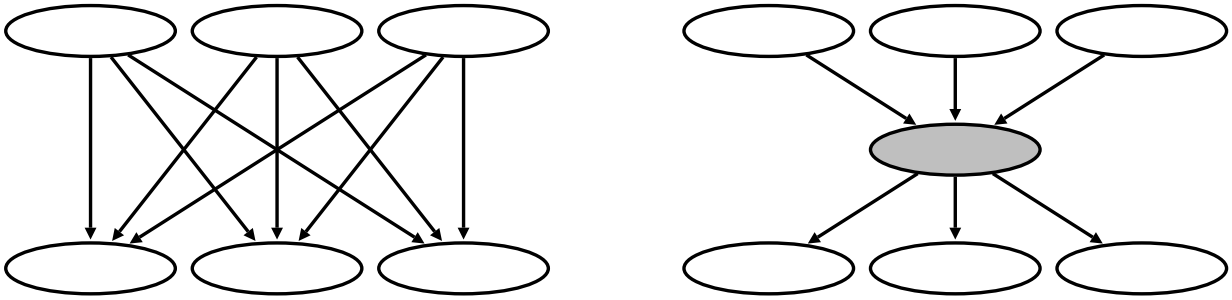


FIG. I.7: Exemple de différence de structures causales avec et sans variable cachée

Quand on veut construire un modèle pour un système mal connu, la présence de variables cachées doit être prise en compte pour construire des modèles utiles, ou plus simples. De la même façon, dans un MDP, on peut aussi considérer l'état du système comme caché, et une variable observable dépendant de l'état caché. C'est le cadre des processus décisionnels de Markov partiellement observables (POMDP) [Russell et Norvig, 2003b]. Dans ce cas, le calcul de politique se base sur des probabilités d'être dans tel ou tel état réel, et ainsi la décision se fait dans cet espace de probabilités, qui est continu. Un POMDP peut être considéré comme un MDP à état continu, et la complexité de résolution de tels problèmes est très grande.

Théorie de l'Information

Dans cette section, nous allons présenter quelques notions issues de la théorie de l'information. En effet, nous allons travailler avec des variables aléatoires liées de manière probabiliste, et la question de l'indépendance de deux variables est très importante.

L'entropie d'un événement pour une variable aléatoire est une façon de quantifier la surprise de la réalisation d'un événement. Plus l'événement est improbable, plus la surprise est importante :

$$h(z_i) = -log[P(z_i)] \tag{I.1}$$

L'entropie H d'une variable aléatoire z se définit par la moyenne sur les différents évènements possibles :

$$H(z) = -\sum_{i=1}^{N} P(z_i) log[P(z_i)] \tag{I.2}$$

Si tous les évènements z_i sont équiprobables, alors on a $H(z) = log(N)$, et la fonction H atteint un maximum. À l'opposé, si un seul des évènements est possible, alors $H(z) = 0$, qui est sa valeur minimale.

On définit également l'entropie jointe sur deux variables x et y qui sera l'information fournie par la mesure simultanée de deux variables :

$$H(x, y) = -\sum_{i,j} P(x_i, y_j) log[P(x_i, y_j)] \tag{I.3}$$

Pour deux variables aléatoires réelles x et y, l'information mutuelle I se définit par :

$$I(x, y) = H(x) + H(y) - H(x, y) \tag{I.4}$$

$$= \sum_{i,j} P(x_i, y_j) log \frac{P(x_i, y_j)}{P(x_i)P(y_j)} \tag{I.5}$$

On peut donc interpréter l'information mutuelle comme la quantité d'information sur une variable donnée par la mesure de l'autre. Pour deux variables indépendantes, on a $I(x, y) = 0$.

Chapitre II

Modèles Stochastiques à État Discret

Dans ce chapitre, notre but est de modéliser sous forme stochastique un processus complexe, et nous prendrons comme exemple la navigation autonome d'un robot mobile. En effet, pour des systèmes réels et pour la plupart des applications, on ne dispose pas en général de modèle a priori de la tâche, car elle est construite dans un but de robustesse opératoire et non en vue de fournir un modèle de son utilisation. Il est alors nécessaire de modéliser sous forme de "boîte noire" une telle tâche, c'est-à-dire construire un modèle après-coup, et non pas en composant des modèles de plus bas niveau, qui sont indisponibles. Lorsque le processus est en interaction forte avec l'environnement, un modèle de cette tâche devra être intrinsèquement stochastique, de façon à capturer les aléas possibles.

Les contributions de ce chapitre sont de plusieurs ordres : tout d'abord est mis en place une modélisation pour un processus peu défini sur un robot réel, puis ce chapitre propose une méthode originale pour déterminer entièrement le modèle, des données brutes issues du processus à la définition de l'espace d'états. Enfin nous mentionnerons comment évaluer un tel modèle.

Les travaux présentés dans ce chapitre (en particulier les sections II.2 sur le traitement des données, II.4 sur la détermination des états cachés et II.6, l'application elle-même) ont été effectués en collaboration étroite avec le Pr. Maria Fox et sous sa direction lors de son séjour sabbatique au LAAS pendant la première année de ma thèse. Le professeur Maria Fox a implémenté les algorithmes d'apprentissage et de classification (i.e. EM et Kohonen). L'auteur a, quant à lui, mis en oeuvre les aspects expérimentaux sur le robot. Ces travaux font l'objet de l'article [Fox, Ghallab, Infantes et Long, 2006], et ce chapitre en reprend de nombreuses formulations et résultats.

Dans un premier temps, nous détaillerons ce qu'est un modèle de Markov caché, puis parlerons du traitement des données brutes en vue d'extraire les informations pertinentes pour le modèle. Nous expliquerons ensuite en détail comment un tel modèle peut être appris à partir d'expérimentations. Suivrons des remarques plus générales sur la forme et les choix du modèle. Nous détaillerons ensuite plusieurs applications possibles de ce type de modèle stochastique et enfin montrerons les résultats obtenus sur notre tâche de navigation autonome.

II.1 Modèle de Markov Caché

Un modèle de Markov caché [Rabiner, 1989; Fox et al., 2006] est constitué d'une variable cachée ou interne représentant l'état du système à modéliser. Cet état interne n'est pas observable directement. En revanche, une autre variable est observable, et est conséquence de cet état interne. Cette variable est appellée observation, comme montré sur la figure II.1. Les liens causaux symbolisés par des flèches sont de nature probabiliste. Ils représentent la probabilité d'avoir les valeurs de la variable d'arrivée sachant la variable de départ. Avec une telle structure, on capture donc à la fois la nature stochastique (bruit) des observations fonction de l'état du système, mais on dispose aussi d'un modèle de transition stochastique pour décrire la dynamique même du système.

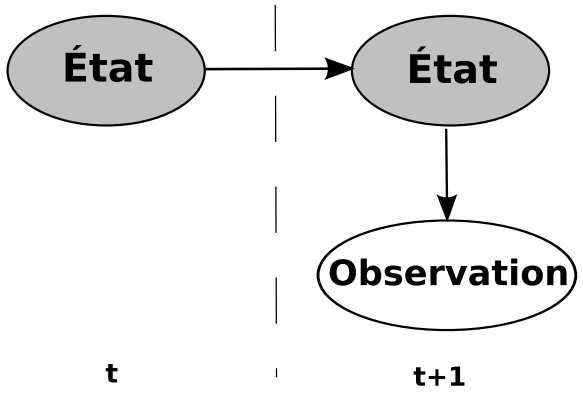


FIG. II.1: Schéma général d'un modèle de Markov caché

Ce type de modèle nécessite que le processus réponde à l'hypothèse Markovienne d'ordre 1 : l'état à un instant t ne dépend que de l'état à l'instant $t-1$, ce qui est généralement peu réaliste. Il existe aussi des extensions de ces modèles pour des hypothèses Markoviennes d'ordre supérieur, mais leur complexité les rend souvent inutilisables. Nous verrons dans les sections II.2 et II.4 comment contourner l'hypothèse Markovienne d'ordre 1 tout en restant dans le cadre des HMM d'ordre 1.

Définition

Formellement, un modèle de Markov caché à observation discrète est constitué des éléments suivants :

1. N, le nombre d'états cachés possibles ; à partir de maintenant nous parlerons simplement d'état et pas d'état caché, c'est-à-dire le nombre de valeurs que peut prendre la variable d'état. On note les différentes valeurs possibles de cette variable $S = \{S_1, S_2, \ldots, S_N\}$. On note également q_t la valeur de la variable d'état à l'instant t ;

2. M, le nombre de symboles d'observations possibles. Les symboles sont notés $V = \{v_1, v_2, \ldots, v_M\}$;
3. la distribution de probabilité de la transition d'état $A = \{a_{ij}\}$ avec :

$$a_{ij} = P(q_{t+1} = S_j | q_t = S_i) \qquad 1 \leq i, j \leq N \tag{II.1}$$

On notera plus simplement :

$$a_{ij} = P(q_{j;t+1} | q_{i;t}) \tag{II.2}$$

4. les distributions de probabilité des observations en l'état j, $B = \{b_j(k)\}$, avec

$$b_j(k) = P(O_t = v_k | q_t = S_j) \qquad \forall t \qquad 1 \leq j \leq N \qquad 1 \leq k \leq M \tag{II.3}$$

avec O_t la valeur de la variable observation à l'instant t, et cette valeur est indépendante de t ; et on notera pour simplifier :

$$b_j(k) = P(O_{kt} | q_{j;t}) \tag{II.4}$$

Lorsque l'on disposera d'une séquence d'observations $O_1 : O_T$ instanciée (par exemple issue d'un processus à modéliser), et qu'il n'y aura pas de confusion possible, on notera simplement $b_j(O_t)$ et alors v_k sera la valeur de l'observation à l'instant t.

5. la distribution l'état initial $\pi = \{\pi_i\}$, avec

$$\pi_i = P(q_1 = S_i) = P(q_{i;1}) \qquad 1 \leq i \leq N \tag{II.5}$$

On notera $\lambda = (A, B, \pi)$ le modèle de Markov caché.

Il est tout-à-fait possible de généraliser ces définitions pour une observation continue (i.e. $k \in C \subset \mathbb{R}$) . Dans ce cas on n'a plus un nombre de symboles d'observations, et $B = \{b_j(k)\}$ sera alors une densité de probabilité.

Étant données les valeurs de N, M, A, B, et π, le HMM peut générer une séquence d'observation

$$O = O_1 O_2 \ldots O_T = O_{1:T} \tag{II.6}$$

comme ceci :

1. choisir une estimation de l'état initial $q_1 = S_i$ en fonction de la distribution initiale sur l'état π ;
2. initialiser t à 1 ;
3. choisir $O_t = v_k$ en suivant la distribution de probabilité sur les observations en l'état S_i, $b_i(k)$;
4. faire évoluer la variable d'état $q_{t+1} = S_j$ en fonction de la distribution de probabilités de transition en l'état S_i, a_{ij} ;
5. faire $t = t + 1$ et retourner à l'étape 3 tant que $t < T$.

De cette façon, on modélise le processus tel une « boîte noire »avec un état interne caché, et un voyant sur cette boite représentant l'observation. De plus, l'observation ne dépend que de l'état courant, et non de l'historique des états traversés. Formellement, on a pour toutes instanciations σ des $q_{1:t}$ se terminant en $q_t = S_j$:

$$P(O_{kt}|q_{1:t} = \sigma, O_{1:t-1}) = P(O_{kt}|q_{j;t}) \tag{II.7}$$

De plus, l'hypothèse Markovienne s'écrit :

$$P(q_{i;t+1}|q_{1:t} = \sigma) = P(q_{i;t+1}|q_{j;t}) \tag{II.8}$$

Problèmes classiques

Ce type de modèle peut être utilisé de plusieurs façons pour traiter des problèmes réels :

1. l'estimation : étant donnée une séquence d'observations $O = O_{1:T}$ et un modèle $\lambda = (A, B, \pi)$, comment peut-on calculer efficacement la probabilité $P(O|\lambda)$ de l'apparition de cette séquence O connaissant le modèle λ ?
2. l'explication : étant donnée une séquence d'observations $O = O_{1:T}$ et un modèle $\lambda = (A, B, \pi)$, quelle est la séquence d'états $Q = q_1 q_2 \dots q_T$ qui explique le mieux l'observation ?
3. l'apprentissage : quel est le modèle $\lambda = (A, B, \pi)$ qui explique le mieux une séquence d'observation O, c'est-à-dire qui maximise $P(O|\lambda)$? En effet, la probabilité d'apparition d'une séquence d'observations n'est pas la même suivant les probabilités de transitions d'état A et les probabilités B d'apparition des symboles d'observations. On va donc chercher les distributions de probabilités qui rendent la probabilité d'apparition d'une vraie séquence observée du système la plus grande possible.

Comme nous allons le voir dans la section II.3, répondre aux problèmes de l'estimation et de l'explication est nécessaire pour traiter l'apprentissage.

Ces trois problèmes de « bas niveaux »nous permettront dans la section II.5 d'envisager d'autres applications, telles la reconnaissance de processus (à travers le premier problème) et le diagnostic (en étendant le deuxième problème).

Exemple

Pour notre application robotique, nous cherchons à modéliser le comportement du robot lors de la navigation. Cet exemple est typique d'un processus de contrôle construit pour la robustesse, mais dont aucun modèle n'existe. Or, disposer d'un modèle de cette tâche permet de la superviser finement, en prévoyant des cas d'échecs à l'avance, ou encore en prédisant les temps ou autres ressources nécessaires. Comme nous l'avons dit, nous disposons de beaucoup d'informations sur l'organisation et les algorithmes internes du robot, mais leur grande complexité nous fait préférer une approche de type « boîte noire ». Typiquement, nous caractérisons le comportement interne par des valeurs de l'état de type qualitatif.

Au vu de la façon dont se comporte le robot, un observateur peut reconnaître différents types de comportements très différenciés au niveau macroscopique, et qui vont influencer grandement la supervision d'un tel processus. Les comportements internes que nous cherchons à reconnaître sont les suivants :

$$S = \{\text{début}, \text{succès}, \text{échec}, \text{hésitation}, \text{évitement}, \text{progrès}, \text{recherche}\} \tag{II.9}$$

Ils constituent notre ensemble de valeurs pour la variable « état interne », comme illustré sur la figure II.2. Ces valeurs sont choisies comme très significatives de différents comportements, tant du point de vue de leur désirabilité que de leur influence sur la tâche globale (vitesse d'exécution, possibilité intuitive de conduire à un échec etc).

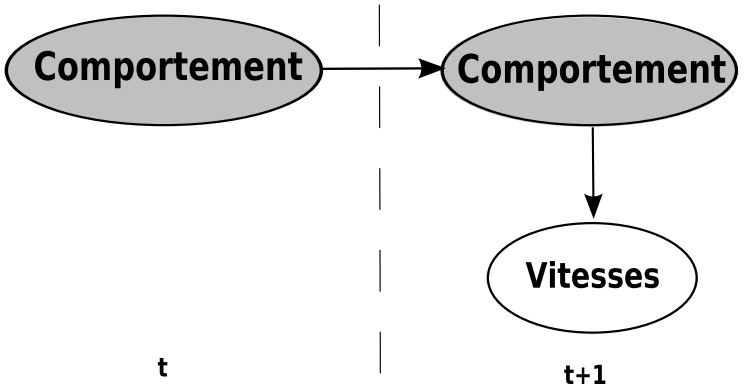


FIG. II.2: Modèle de Markov caché pour la navigation autonome d'un robot

Au niveau des variables observables, le processus de contrôle nous donne de nombreuses informations. Pour l'instant nous disposons entre autres des vitesses linéaires et angulaires instantanées telles que données en consigne au robot. Elles constituent les données brutes. Il faut ensuite passer aux observations discrètes du modèle ; on peut sans entrer dans le détail imaginer les classifier (voir section II.2) en 3 classes pour la vitesse angulaire (avant, droite, gauche) et autant pour la vitesse linéaire (rapide, lent, arrêt) par exemple. Dans ce cas, la variable d'observation peut prendre les valeurs :

$$V = \{(\text{avant}; \text{rapide}), (\text{avant}; \text{lent}), (\text{avant}; \text{arrêt}), (\text{gauche}; \text{rapide}), \ldots\} \tag{II.10}$$

On obtient alors une première ébauche de HMM très simple.

II.2 Pré-traitement des Données

Dans cette section, nous allons voir comment traiter des données brutes issues d'un processus réel de façon à passer aux symboles d'observations tel qu'utilisés par un modèle de Markov caché. En effet, en général, l'espace des données brutes est de taille infinie (les données sont en général multiples et de nature continue), alors que l'espace des observations du modèle est discret et fini. Se pose alors la question du passage de cet espace continu potentiellement de grande dimension à un espace fini.

Sélection

Suivant les applications visées, on peut disposer d'un nombre très variable de données brutes sur le processus à modéliser. On peut aussi bien disposer d'une simple indication booléenne (cas d'un voyant allumé ou éteint) que d'une multitude de données capteur dans le cadre d'une application robotique comme la nôtre.

Généralement, les données brutes sont très bruitées, et certains pics non significatifs peuvent apparaître. Ainsi, un pré-traitement sous forme de lissage est souvent nécessaire. Mais ce lissage peut être vu autrement. En effet, dans notre exemple de navigation robotique autonome, nous utilisons pour l'instant les consignes en vitesses instantanées comme observations, qui peuvent être fortement discontinues du fait de la nature réactive et sans mémoire du contrôleur de navigation que nous avons utilisé. Plus généralement, il faut garder en tête que l'espace des observations est de nature plus élaborée que les données brutes issues des capteurs. On peut également penser qu'il peut être judicieux que ce système simple soit mieux modélisé en violant l'hypothèse markovienne : l'observation ne devrait pas être une mesure instantanée, mais être une fonction d'un certain historique. En effet, un comportement d'hésitation du robot n'est pas lié à une vitesse angulaire grande ou petite, mais à une succession de gauche-droite, par exemple. Une telle phase de pré-traitement éventuellement sur une fenêtre temporelle porte le nom de sélection de caractéristiques (voir [Fox et al., 2006]). Chaque vecteur de données brutes est de grande dimension suivant tout ce qui peut être enregistré (exemple des vitesses instantanée, mais aussi données sur l'environnement courant, ou encore paramètres divers de la navigation). On cherche à extraire de ces données brutes des données traitées qui soient plus caractéristiques du processus, et surtout plus discriminantes. Un exemple simple peut être que les données brutes contiennent des vitesses, mais que l'on pense que ce sont les accélérations qui sont caractéristiques, et non les valeurs des vitesses.

Le principe général consiste à prendre une sous-ensemble des données brutes par exemple sur un intervalle de temps, et à extraire les variables caractéristiques qui paraissent utiles pour modéliser le processus. On peut voir sur la figure II.3 le principe général, appelé *fenêtre glissante* qui permet de passer des données brutes aux caractéristiques utiles à la modélisation.

Formellement, on définit un ensemble de k fonctions sensorielles :

$$f_i : \text{processus à modéliser} \times \text{environnement} \times \text{temps} \rightarrow \gamma_i \qquad 1 \leq i \leq k \tag{II.11}$$

avec $\gamma_i \subseteq \mathbb{R}$ est l'espace possible pour une donnée brute. On a alors un espace de données brutes de dimension k :

$$\Phi = \gamma_1 \times \gamma_2 \times \ldots \times \gamma_k \tag{II.12}$$

On cherche ensuite à passer de cet espace Φ à l'espace des caractéristiques choisies Ψ par la fonction g. On peut éventuellement prendre L points dans Φ pour obtenir un point dans Ψ :

$$g : \Phi^L \rightarrow \Psi \tag{II.13}$$

La définition de cet espace Ψ et surtout trouver la fonction g sont des étapes critiques, telles qu'illustrées sur la figure II.3 par le traitement de chaque séquence. On peut effectuer un lissage sur L points en vue de supprimer des points particuliers et/ou effectuer un filtrage en vue de ne

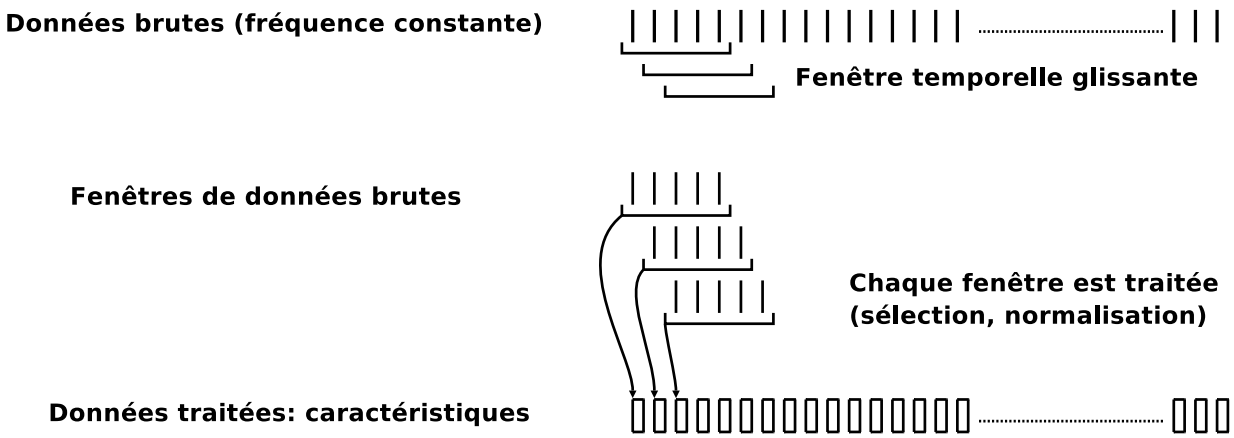


FIG. II.3: Principe de la fenêtre glissante

sélectionner que ce qui nous intéresse. On peut extraire la moyenne des vitesses instantanées sur la fenêtre par exemple, mais aussi le minimum, le maximum, une somme pondérée, la différence entre le dernier point et le premier ou encore une somme de valeurs absolues des différences entre chaque point et son précédent, pour évaluer l'ensemble des variations sur la fenêtre. Tout type de fonction de traitement est potentiellement intéressant, et cette phase de traitement des données est critique pour la modélisation du processus. Ici une phase d'ingénierie avec ou par un expert du processus à modéliser et indispensable car travailler directement sur les données brutes peut conduire à des modèles de qualité très médiocre.

Exemple

La navigation autonome utilisée se base sur une implémentation de l'algorithme *Nearness Diagram* [Minguez, Osuna et Montano, 2004]. Malgré le fait que nous ayons accès à l'implémentation du processus, une navigation autonome basée sur ce principe reste très dure à modéliser par une approche « bottom-up »du fait de sa grande complexité.

Pour notre application robotique, les données brutes sont mesurées à une fréquence de 5 Hertz et sont composées des éléments suivants :

- les coordonnées du robot (dans un repère local) ;
- les vitesses instantanées de commande (vitesses linéaires et angulaires) ;
- la distance au but (estimée comme une distance euclidienne) ;
- l'encombrement autour du robot défini comme une somme sur les distances d_i par couloirs devant le robot pondérés par des poids w_i gaussiens tels qu'un poids maximal est donné devant le robot, et plus faible sur ses côtés (voir figure II.4). Les distances aux obstacles sont lues depuis le télémètre laser qui renvoie pour chaque secteur angulaire d'un demi-degré la distance à l'obstacle le plus proche.

Les caractéristiques sont sélectionnées par lissage et filtrage sur une fenêtre glissante de 24 données brutes consécutives dans le temps (ce qui correspond à peu près à 5 secondes de temps réel, et qui reflète assez bien la dynamique du système) :

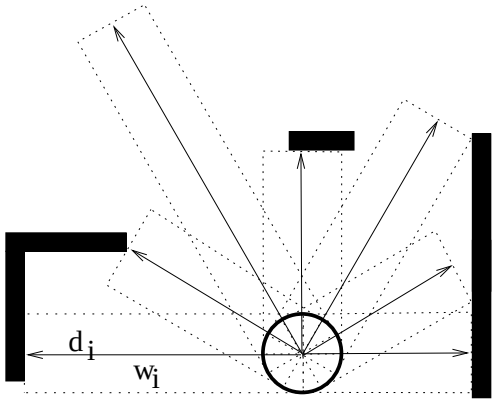


FIG. II.4: Schéma de l'encombrement autour du robot

1. distance depuis le point de départ (moyenne sur la fenêtre déduit des coordonnées de chaque point de mesure) ;
2. distance au but (moyenne sur la fenêtre de la donnée brute correspondante) ;
3. distance curviligne parcourue sur la fenêtre, définie comme une somme des distances parcourue entre chaque point de mesure : $\sum \Delta(x_i, y_i)$;
4. changement de cap, calculé comme la différence entre l'orientation du robot en fin de fenêtre temporelle et celle en début de cette fenêtre ;
5. rotation totale (somme des changements de cap), calculé comme une somme des valeurs absolues des changements d'orientation du robot entre les différentes points de mesure, de façon à caractériser des oscillations du robot lors d'un certain laps de temps ;
6. encombrement moyen : la moyenne sur la fenêtre de la valeur d'encombrement autour du robot ;
7. vitesse linéaire moyenne sur la fenêtre ;
8. accélération : différence sur la fenêtre entre la vitesse linéaire la plus élevée et la plus faible.

Classifications

Lorsque l'on dispose de nos points de l'espace Ψ, appelés caractéristiques, il reste à passer dans l'espace des observations V tel que défini pour un modèle de Markov caché. Pour passer de cet espace à un ensemble de symboles d'observation, on utilise une technique de classification, qui est de la forme :

$$classif : \Psi \rightarrow V \tag{II.14}$$

Une des techniques les plus simples est l'algorithme appelé *k-means* (voir [Kanungo et al., 2002]), dans lequel on fixe à l'avance le nombre de classes (classification supervisée). Étant données les valeurs minimales, maximales et le nombre de classes, l'algorithme définit des bornes entre classes a priori.

1. Il distribue les points de Ψ à classifier dans ces classes en fonction de ces bornes.
2. Ensuite, il calcule la moyenne de ces points dans chaque classe,
3. et les nouvelles bornes inter-classes sont à mi-distance entre les moyennes.

Il reprend ensuite au début en distribuant de nouveau les vecteurs dans les nouvelles classes, recalculant les moyennes par classes et ainsi de suite. On arrête ce processus au bout d'un nombre donné d'itérations, mais en général les bornes se stabilisent extrêmement vite, c'est-à-dire en quelques itérations. Une variante courante consiste à prendre la médiane et non la moyenne dans chaque classe. Cet algorithme est fait pour travailler sur des vecteurs unidimensionnels, mais on peut quand même l'utiliser dans notre cas en généralisant en utilisant une distance euclidienne sur nos vecteurs à m dimensions, qui donne un poids égal à chacune des m composantes.

Dans le cas d'observations continues, on approxime la fonction continue d'observations par une somme de gaussiennes. Mais là encore, fixer le nombre de gaussiennes pour l'approximation de la fonction continue doit être fait à l'avance.

Lorsque le processus est plus complexe, on ne connaît pas précisément le nombre de classes, mais on a juste un ordre de grandeur de celui-ci (ou alors on se fixe une limite de façon à ce que l'apprentissage du modèle soit possible en temps raisonnable). Dans ce cas, il faut avoir recours à des algorithmes de classification non-supervisés.

Une technique de choix est l'utilisation de cartes de Kohonen [Kohonen, 1984]. Nous allons détailler le principe d'un classifieur de Kohonen bi-dimensionnel. (figure II.5).

Soient $d_i \in \Psi$ les vecteurs à classifier, ils sont de dimension r. On définit un espace Υ tel que $\Psi \subset \Upsilon$, le plus simple est de prendre $\Upsilon = \mathbb{R}^r$. On prend un ensemble de vecteurs aléatoires $e_j \in \Upsilon$. Ces vecteurs sont répartis aléatoirement dans une grille bi-dimensionnelle de dimension $s \times s$. On définit grâce à cette grille un voisinage dans Υ : deux vecteurs sont voisins s'ils sont dans deux cases contiguës.

Une phase d'auto-organisation se décompose comme suit :

1. on prend un des vecteurs à classifier d_i ;
2. on trouve dans les Υ celui qui en est le plus proche : e_j (suivant une distance euclidienne par exemple)
3. ce vecteur le plus proche devient $e_j \leftarrow e_j + \eta \times d_i$ (il se rapproche de celui à classifier), avec $\eta < 1$;
4. les vecteurs f_k du voisinage de e_j deviennent $f_k \leftarrow f_k + \eta' d_i$, avec $\eta' < \eta$ (ils se rapprochent aussi du vecteur à classifier, mais moins que e_j) ;
5. on passe au vecteur d_{i+1} et ainsi de suite...

Cette phase d'auto-organisation doit être réitérée un grand nombre de fois en fonction du nombre de vecteurs d_i avant d'atteindre un point fixe (c'est-à-dire que les vecteurs e_j ne varient plus). On obtient alors en sortie l'ensemble des vecteurs e_j modifiés en fonction des vecteurs à classifier d_i. On dit alors que la carte s'est organisée. En effet, les vecteurs de cases proches sont proches entre eux, et on a donc une relative continuité lorsque l'on suit un voisinage. On constate également que les vecteurs aléatoires e_j de départ ont disparu, et que les cases contiennent toutes des vecteurs de d_i (grâce au grand nombre d'itérations de la phase d'organisation).

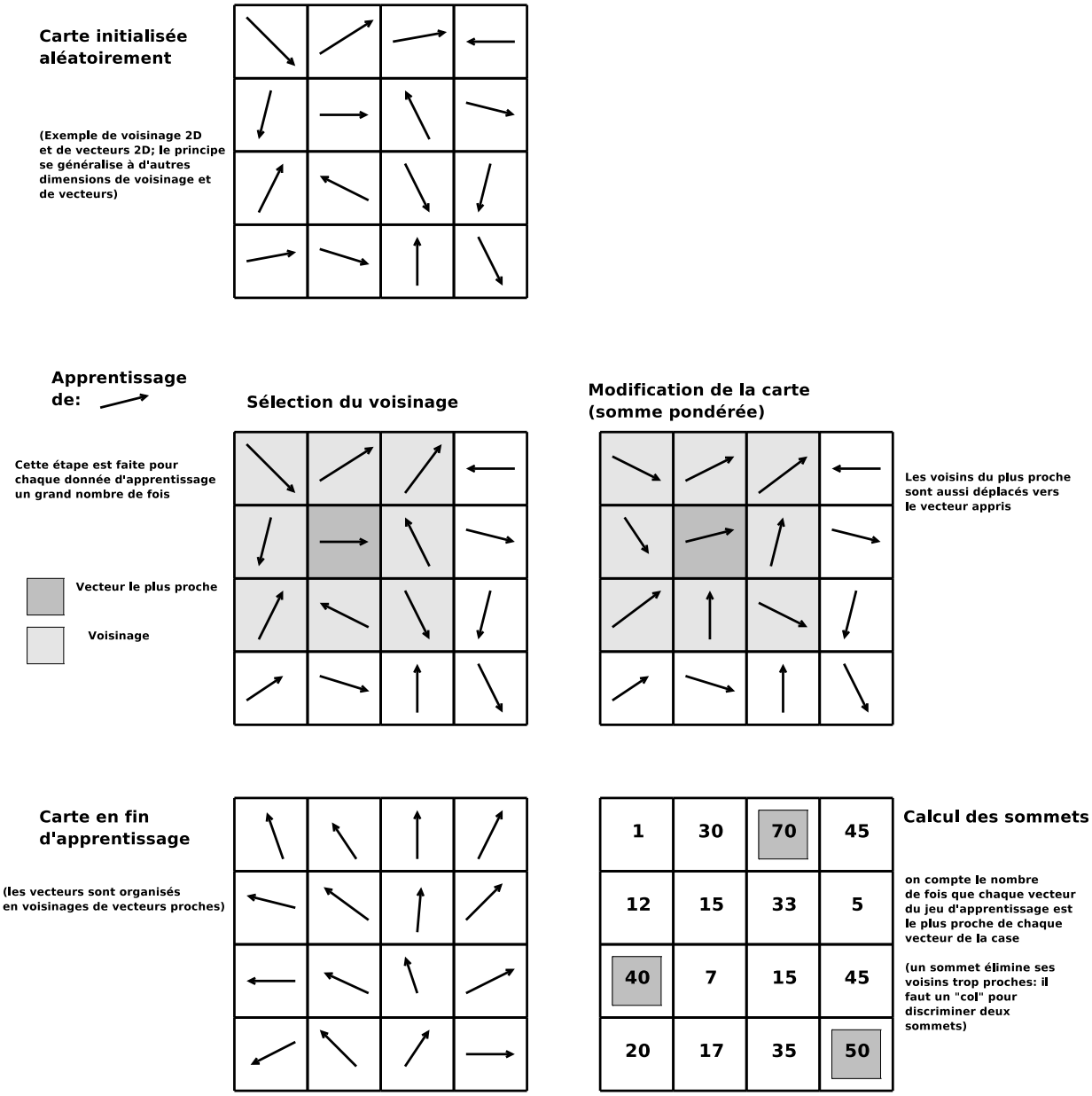


FIG. II.5: Carte de Kohonen pour la classification

Une fois que la carte est organisée, il nous reste à trouver les classes elles-mêmes. Pour cela, on parcourt les vecteurs d_i une ultime fois et pour chacun d'entre eux on regarde la case la plus proche. Cette case marque alors un point. On va ensuite regarder le score de chaque case et construire le paysage correspondant en se servant de ce score comme altitude. Il faut alors noter que la largeur de la grille est un facteur déterminant pour que ce processus se passe bien, car si on a des scores nuls, le paysage construit perd tout son sens. La taille de la grille doit être au moins 500 fois plus petite que $card\{d_i\}$ [Kohonen, 1984].

Les centres des classes seront alors les sommets de ce paysage (certains sommets sont éliminés s'ils sont trop proches d'un autre qui est beaucoup plus haut qu'eux-mêmes). Chaque classe est ainsi définie par le vecteur sommet, qui est appelé vecteur caractérisant de la classe. Pour chaque vecteur de caractéristiques, on détermine à quelle classe il appartient par un simple calcul de distance aux différents sommets sélectionnés (aux différents vecteurs caractérisants).

Exemple

Dans notre exemple, nous avions des vecteurs de caractéristiques à huit dimensions après traitement et sélection. Il convient ensuite de projeter ces vecteurs à huit dimensions dans l'espace des symboles d'observations. Ceci est fait grâce à une classification de Kohonen.

L'algorithme utilisé est exactement celui décrit précédemment. Nous avons utilisé un grille bi-dimensionnelle avec 4 voisins par case. La dimension de la grille doit être au moins 500 fois plus petite que la nombre de vecteurs de données à classifier [Kohonen, 1984]. Comme nous avons autour de 15 000 vecteurs de données brutes, nous avons utilisé des dimensions entre 15 et 45 (pour la longueur d'un coté). Des grandes tailles introduisent plus de bruit dans la classification (avec de nombreuses classes peu discriminantes), alors que des petites tailles induisent un trop faible facteur de discrimination. Une dimension de 25 semble optimale dans notre cas.

Cette technique est assez simple d'implémentation, et surtout permet de ne pas fixer le nombre de classes à l'avance. Si l'on ne dispose pas de beaucoup d'informations sur les données à classer, et en particulier sur les liens entre les composantes de ces données (comme des covariances par exemple), alors cette technique permet quand même de faire une classification robuste qui va se baser sur les composantes les plus discriminantes des vecteurs à classifier. En regardant les vecteurs caractérisants des classes, on peut voir quelles sont les composantes qui ont été significatives pour la classification, ce qui est d'une grande aide lorsque les données sont trop complexes pour être étudiées à la main.

II.3 Apprentissage Quantitatif du modèle

Lorsque l'on a défini le nombre d'états et celui des observations, que l'on dispose de séquences d'observations O réelles, on peut s'intéresser au troisième problème de la section II.1, l'apprentissage d'un modèle λ qui maximise $P(O|\lambda)$, à savoir qui explique le mieux possible cette séquence d'observations.

L'algorithme utilisé pour cela s'appelle *Expectation-Maximization*. Il consiste en une recherche locale autour d'un modèle de départ en mettant à jour les probabilités de transitions et d'observations. Il est connu pour atteindre un maximum local de $P(O|\lambda)$ (voir [Fox et al., 2006; Rabiner, 1989]).

Expectation-Maximization

On appelle

$$O = O_1, O_2, \ldots, O_T = O_{1:T} \tag{II.15}$$

la séquence d'observations, et

$$\lambda = (A, B, \pi) \tag{II.16}$$

le modèle de Markov caché. On va donc chercher à mettre-à-jour $A = \{a_{ij}\}$ et $B = \{b_j(k)\}$ pour maximiser $P(O|\lambda)$.

Pour cela on définit les probabilités d'être en chaque état à chaque pas de temps* :

$$\gamma_t(i) = P(q_{i;t}|O, \lambda) \tag{II.17}$$

Et de la même façon, les probabilités de transitions sont :

$$\xi_t(i, j) = P(q_{i;t}, q_{j;t+1}|O, \lambda) \tag{II.18}$$

La démarche est alors la suivante : lorsque les γ et ξ sont connus (en fonction de A et B), on pourra alors maximiser $P(O|\lambda)$ en calculant les nouveaux a_{ij} en disant que la probabilité de faire une transition de S_i vers S_j est la somme des transitions supposées de S_i vers S_j suivant la séquence O normalisée (divisée) par la probabilité attendue d'être en S_i. De la même façon, on pourra calculer les nouveaux $b_j(k)$ en divisant le nombre de fois dans lequel on pense être dans l'état S_j et où l'on voit v_k par le nombre de fois on l'on pense être dans l'état S_j. On peut également mettre à jour les probabilités a priori comme la probabilité d'être en l'état S_i en $t = 1$. Lorsque l'on a un nouveau modèle, on peut recalculer les valeurs de $\gamma_t(i)$ et $\xi_t(i)$ et refaire une phase de maximisation et ainsi de suite. On itère jusqu'à ce que le modèle n'améliore plus $P(O|\lambda)$, et alors on est dans un maximum local.

L'algorithme est alors le suivant :

Tant que $P(O|\lambda)$ s'améliore (augmente) :

1. calculer les $\gamma_t(i)$ et les $\xi_t(i, j)$ pour tous i, j, t en fonction de $\lambda = (A, B, \pi)$;
2. $a_{ij} \leftarrow \frac{\sum_t \xi_t(i,j)}{\sum_t \gamma_t(i)}$
3. $b_j(k) \leftarrow \frac{\sum_{t, O_t = v_k} \gamma_t(j)}{\sum_t \gamma_t(j)}$
4. $\pi_i \leftarrow \gamma_1(t)$
5. $\lambda \leftarrow (\{a_{ij}\}, \{b_j(k)\}, \{\pi_i\})$
6. recalculer $P(O|\lambda)$

La difficulté consiste à calculer efficacement les $\gamma_t(i)$ et les $\xi_t(i, j)$. Pour cela, on a recours à l'algorithme récursif *forward-backward*.

*On remarque que calculer ces valeurs correspond exactement à répondre au deuxième problème standard des HMM

Forward-Backward

Message en avant

On définit la probabilité d'arriver en un état S_i connaissant le modèle et le début d'une séquence d'observations $O_{1:t} = O_1, O_2, \ldots, O_t$:

$$\nu_t(i) = P(q_{i;t}|O_{1:t}, \lambda) \tag{II.19}$$

On définit un message en avant comme la probabilité d'être en S_i à l'instant t connaissant le modèle et le début de la séquence d'observations pondéré par la probabilité de voir le début de l'observation :

$$\begin{aligned} \alpha_t(i) &= \nu_t(i)P(O_{1:t}) && \text{(II.20)} \\ \alpha_t(i) &= P(q_{i;t}|O_{1:t}, \lambda)P(O_{1:t}) && \text{(II.21)} \\ \alpha_t(i) &= P(q_{i;t}, O_{1:t}|\lambda) && \text{(II.22)} \end{aligned}$$

On peut alors calculer récursivement ce message en avant en fonction de ses prédécesseurs :

1. initialisation :
$$\alpha_1(i) = \pi_i b_i(O_1) \tag{II.23}$$
2. induction :
$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i)a_{ij}]b_j(O_{t+1}) \tag{II.24}$$
3. terminaison[†] :
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{II.25}$$

Message arrière

De la même façon, on peut définir un message arrière comme la probabilité de faire la fin $O_{t+1:T} = O_{t+1}, \ldots, O_T$ d'une séquence d'observation en partant de l'état S_i en l'instant t :

$$\beta_t(i) = P(O_{t+1:T}|q_{i;t}, \lambda) \tag{II.26}$$

Ce message peut également être calculé récursivement à partir de ses successeurs, en partant de la fin de l'observation (instant T) :

1. initialisation :
$$\beta_T(i) = 1 \qquad \forall i \tag{II.27}$$
2. induction :
$$\beta_t(i) = \sum_j a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \tag{II.28}$$

[†]On remarque ici que cette équation permet de répondre au premier problème classique des HMM.

Expectation

Lorsque l'on a calculé tous les $\alpha_t(i)$ et les $\beta_t(i)$, nous obtenons :

$$\begin{aligned} \gamma_t(i) &= P(q_{i;t}|O_{1:t},\lambda)P(O_{t+1:T}|q_{i;t},\lambda) && \text{(II.29)} \\ &= \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} && \text{(II.30)} \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} && \text{(II.31)} \end{aligned}$$

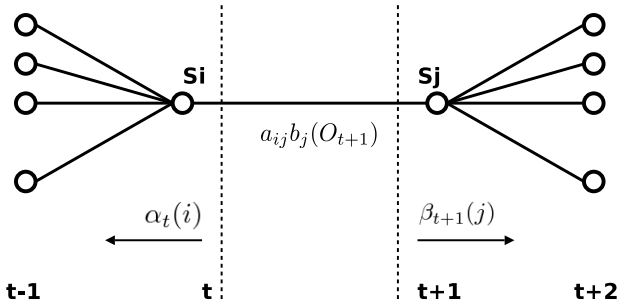


FIG. II.6: Illustration du calcul nécessaire pour obtenir $\xi_t(i,j)$

Pour obtenir $\xi_t(i,j)$ on peut déduire de la figure II.6 l'expression correspondante :

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \quad \text{(II.32)}$$

Or :

$$\xi(i,j) = \sum_{t=1}^{T}\xi_t(i,j) \quad \text{(II.33)}$$

On a donc tous les éléments pour mettre à jour le modèle.

Remarques sur la complexité

La complexité de la phase d'induction pour une variable et un pas de temps est donc en $O(NM)$, avec N la taille de l'espace d'état (l'arité de la variable cachée) et M le nombre d'observations possibles, cela si l'on considère que le temps de recherche/calcul de $b_j(O_{t+1})$ est en

$O(M)$ et que les temps de recherche de a_{ij} et des précédents $\alpha_t(i)$ sont en $O(1)$. Ceci est irréaliste pour une implémentation classique, l'usage d'une structure de donnée particulièrement bien adaptée est nécessaire pour obtenir ce genre de performances, et même ainsi, la recherche des a_{ij} est au moins en $O(N)$. Il faut faire cette phase d'induction pour chaque valeur de l'état, on monte donc en $O(N^2M)$, et ensuite pour chaque pas de temps, l'induction est alors au mieux en $O(TN^2M)$. Même pour une bonne implémentation on arrive en $O(TN^3M)$ suivant la façon de stocker les a_{ij}.

De la même façon que précédemment, la complexité du calcul du message arrière est en $O(TN^2M)$ et généralement en $O(TN^3M)$, encore une fois suivant la façon de stocker les a_{ij}.

La complexité d'une phase de mise à jour des a_{ij} est alors en $O(T^2N^2M)$ au meilleur cas, en utilisant des caches pour les facteurs de normalisation. Comme nous l'avons déjà mentionné, une implémentation sera plutôt en $O(T^2N^3M)$. Il faut alors noter que ces résultats sont pour une seule itération de la mise à jour du modèle, et que ce nombre d'itérations n'est pas fixé a priori.

Implémentation

Il faut remarquer que cet algorithme manipule des probabilités qui tendent vers 0 de façon exponentiellement rapide avec la longueur des observations. Quelle que soit la précision machine utilisée pour l'implémentation, on va forcément dépasser la capacité de représentation des réels. Pour cela, un mécanisme de mise à l'échelle des $\alpha_t(i)$ et $\beta_t(i)$ est nécessaire. Il est clairement expliqué dans [Fox et al., 2006]. Le principe est le suivant : lorsque l'on a calculé touts les $\alpha_t(i)$ pour un t donné suivant l'équation II.24, on les multiplie par un coefficient c_t :

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{II.34}$$

et alors on a :

$$\hat{\alpha}_t(i) = \left(\prod_{\tau=1}^{t} c_\tau \right) \alpha_t(i) \tag{II.35}$$

Ce facteur de mise à l'échelle s'intègre élégamment dans le calcul récursif, en effet, on va calculer récursivement :

$$\alpha_t(i) = \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t) \tag{II.36}$$

Il vient alors l'expression des $\hat{\alpha}_t(i)$:

$$\hat{\alpha}_t(i) = \frac{\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t)} \tag{II.37}$$

$$= \frac{\sum_{j=1}^{N} \left(\prod_{\tau=1}^{t-1} c_\tau\right) \alpha_{t-1}(j) a_{ij} b_j(O_t)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \left(\prod_{\tau=1}^{t-1} c_\tau\right) \alpha_{t-1}(j) a_{ij} b_j(O_t)} \tag{II.38}$$

$$= \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} \tag{II.39}$$

Donc on peut voir qu'en appliquant le mécanisme récursif de calcul des messages avant aux messages mis à l'échelle, cette mise à l'échelle est répercutée telle quelle aux étapes de récursion suivantes. On n'a donc pas besoin d'enlever puis de remettre ce facteur dans la récursion, on peut juste l'appliquer avant un débordement éventuel, et appliquer le calcul récursif classique.

De la même façon, on va appliquer le *même* facteur de mise à l'échelle au message arrière, on aura alors :

$$\hat{\beta}_t(i) = c_t \beta_t(i) \tag{II.40}$$

L'intérêt d'utiliser le même facteur devient évident lorsque l'on regarde la mise à jour du modèle de transition. On peut écrire $\hat{\alpha}_t(i)$ de la façon suivante :

$$\hat{\alpha}_t(i) = \left[\prod_{s=1}^{t} c_s\right] \alpha_t(i) \tag{II.41}$$

$$= C_t \alpha_t(i) \tag{II.42}$$

et de la même façon $\hat{\beta}_{t+1}(j)$ s'écrit :

$$\hat{\beta}_{t+1}(j) = \left[\prod_{s=t+1}^{T} c_s\right] \beta_{t+1}(j) \tag{II.43}$$

$$= D_{t+1} \beta_{t+1}(j) \tag{II.44}$$

Or on a :

$$C_t D_{t+1} = \prod_{s=1}^{t} c_s \prod_{s=t+1}^{T} c_s \quad \text{(II.45)}$$

$$= \prod_{s=1}^{T} c_s \quad \text{(II.46)}$$

$$= C_T \qquad \text{indépendant de t} \quad \text{(II.47)}$$

De cette façon, la mise à jour du modèle devient :

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)} \quad \text{(II.48)}$$

$$= \frac{\sum_{t=1}^{T-1} C_t \alpha_t(i) a_{ij} b_j(O_{t+1}) D_{t+1} \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} C_t \alpha_t(i) a_{ij} b_j(O_{t+1}) D_{t+1} \beta_{t+1}(j)} \quad \text{(II.49)}$$

$$= \frac{C_T \sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{C_T \sum_{t=1}^{T-1} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \quad \text{(II.50)}$$

On constate alors qu'en utilisant ce même facteur c_t pour la mise à l'échelle pour les $\alpha_t(i)$ et $\beta_t(i)$ c'est bien la bonne mise à jour du modèle qui est faite, sans changer l'algorithme.

L'affaire est différente pour le calcul de $P(O|\lambda)$, où l'on utilisait l'équation II.25. En revanche, on peut voir que :

$$\prod_{t=1}^{T} c_t \sum_{i=1}^{N} \alpha_T(i) = C_T \sum_{t=1}^{N} \alpha_T(i)$$

$$= 1$$

D'ou l'on peut déduire les expressions suivantes :

$$\prod_{t=1}^{T} c_t P(O|\lambda) = 1 \tag{II.51}$$

$$P(O|\lambda) = \frac{1}{\prod_{t=1}^{T} c_t} \tag{II.52}$$

$$log[P(O|\lambda)] = -\sum_{t=1}^{T} log(c_t) \tag{II.53}$$

Comme en général la probabilité de voir l'observation est trop faible pour être représentée en machine dès que l'observation est longue, on utilisera l'équation II.53 en mémorisant les facteurs de mise à l'échelle pour calculer cette probabilité. C'est pour cette raison que l'on ne parle pas de "likelihood" de l'observation, mais de sa "log-likelihood".

Dans le cas ou l'on utilise plusieurs séquences d'observations, ce mécanisme sert de plus à pondérer les apports de chaque observation lors de la phase de maximisation. En effet, si l'on a K séquences d'observations $O = [O^{(1)}, O^{(2)}, ..., O^{(K)}]$ avec $O^{(k)} = (O_1^{(k)} O_2^{(k)} ... O_{T_k}^{(k)})$ la k^{ime} séquence d'observation, on va chercher à maximiser :

$$P(O|\lambda) = \prod_{k=1}^{K} P(O^{(k)}|\lambda) \tag{II.54}$$

$$= \prod_{k=1}^{K} P_k \tag{II.55}$$

La mise à jour du modèle va se faire en pondérant les probabilités d'apparition des transitions et des observation par la probabilité de chaque séquence d'observation :

$$a_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \beta_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \tag{II.56}$$

$$b_j(l) = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{\substack{t=1 \\ O_t=v_l}}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)}{\sum_{k=1}^{K} \frac{1}{P_k} \alpha_t^k(i) \beta_t^k(i)} \tag{II.57}$$

Si l'on utilise à la fois des observations multiples et le mécanisme de mise à l'échelle, alors les termes mis à l'échelle sont déjà pondérés par la probabilité d'observation de la séquence, et

alors on a simplement :

$$a_{ij} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) a_{ij} b_j(O_{t+1}^{(k)}) \hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)} \tag{II.58}$$

Malheureusement, dans ce cas, la mise à jour des probabilités a priori sur la variable d'état ne peut plus se faire simplement (voir [Rabiner, 1989]), et donc le choix de probabilités a priori devient critique (voir section II.4).

II.4 Sur la Structure

Nous avons vu comment apprendre un modèle de type HMM, et comment passer des données brutes aux symboles d'observations. Cependant certaines remarques s'imposent au niveau du paramètre de structure principal d'un modèle de Markov caché, à savoir la taille de l'espace d'états.

Taille de l'espace d'état connue

Dans certains cas, la taille de l'espace d'état est connue. Pour certaines applications telles que la reconnaissance de la parole, on sait que ce que l'on cherche à reconnaître (comme le phonème prononcé) est dans un ensemble bien défini. Cependant lorsque l'on s'intéresse à modéliser un processus mal connu, ce n'est pas forcément le cas.

Alors, le sens de cette arité doit plutôt se lire comme « je cherche à apprendre le meilleur modèle de Markov caché *tel que l'arité cachée soit N* ». Dans ce cas, on doit bien avoir en tête qu'un nombre trop grand de valeurs pour la variable d'état ne doit pas en théorie pénaliser $P(O|\lambda)$ car certains états seront simplement soit non-significatifs, c'est-à-dire jamais atteints par le système, soit deux états seront équivalents (auront les mêmes probabilités d'y arriver, les mêmes probabilités pour en sortir et donneront les mêmes observations). Le pire problème est que ce trop grand nombre va pénaliser le temps d'apprentissage du modèle. En revanche, fixer un nombre de valeurs d'état trop petit va conduire à un très mauvais modèle, incapable de discriminer entre certains états très différents.

Se pose également le problème de la sémantique de l'état interne. Si ce que l'on veut reconnaître fait partie de la variable d'état (par exemple l'état échec de la variable comportement de notre robot), le nombre d'états doit refléter ce sens. Or ce nombre d'état peut être mal connu, mais surtout, lorsque l'on va faire du suivi, même si l'on sait que l'on a un état "échec", et un état "succès", on ne sait pas lequel des états cachés correspond à ces valeurs intuitives. Pour l'algorithme d'apprentissage, toutes les valeurs de la variable cachée sont équivalentes, sauf en ce qui concerne les probabilités a priori données comme point de départ pour l'apprentissage.

Ceci nous mène au deuxième problème : comme on effectue une recherche locale qui utilise les probabilités a priori sur la variable d'état, il faut d'une façon ou d'une autre donner de

« bonnes valeurs »à ces probabilités, ainsi qu'aux probabilités de transitions et d'observations qui servent de point de départ à l'apprentissage du modèle. Dans ce cas, choisir un nombre quelconque de valeurs pour la variable d'état peut poser problème. Et comme pour l'algorithme tous les états cachés sont équivalents, leur nombre et les probabilités initiales sont les seuls facteurs discriminants, il faut alors être très prudent en ce qui concerne ces choix.

Comment Choisir ?

Dans cette logique, on pourrait choisir de garder comme nombre d'état simplement le nombre des états que l'on souhaite reconnaître. Mais si l'on donne une valeur trop faible à ce nombre d'états, le pouvoir discriminant de ces états risque de se perdre. En effet, dans notre exemple, on peut avoir plusieurs cas d'hésitation du robot par exemple, un pour un environnement trop encombré et un pour un environnement trop symétrique. Ces deux cas seront très différent, et on ne peut pas forcément se fier à l'intuition de l'expert du processus à modéliser.

Une solution naïve serait d'apprendre un modèle avec un nombre peu élevé d'états, puis augmenter cette arité jusqu'à stabilisation de $P(O|\lambda)$. Si cette solution pourrait être satisfaisante (quoique lourde) d'un point de vue calculatoire, elle n'a aucun sens si l'on souhaite avoir une sémantique pour les différentes valeurs de l'état (sans même parler du choix complètement aléatoire des probabilités a priori sur la variable d'état ni du choix du modèle initial avant apprentissage).

Nous proposons ici un processus qui a pour but d'optimiser le nombre d'états cachés sans faire de recherche exhaustive, tout en gardant un sens aux états. Il a été nommé *state-splitting* en regard de sa façon de fonctionner.

Découpage d'états

Cet algorithme peut être vu comme une méthode semi-supervisée de détermination du nombre d'états.

Cette méthode part du constat que le concepteur du modèle dispose de certaines informations sur le processus complexe à modéliser comme des grandes classes d'états abstraits, mais qu'il n'a pas d'information fine pour trouver à quels états physiques ces classes peuvent bien correspondre. Inversement, un mécanisme automatique peut trouver des classes précises basées sur des données, mais n'a pas connaissance des grandes classes de comportements du système, ni de ce que veut modéliser ou reconnaître le concepteur. On introduit donc le rôle d'un observateur du système qui cherche à modéliser le comportement macroscopique d'un système autonome. Pour la construction automatique du modèle, on dispose d'observations fines (microscopiques) qui résultent du comportement macroscopique, mais l'observateur voit quant à lui le comportement macroscopique global uniquement, sans avoir accès aux données microscopiques. On a donc ici une complémentarité à exploiter.

L'idée va donc consister en un découpage automatique en sous-comportements des comportements reconnus ou désirés par un observateur extérieur. Ces sous-comportements seront au final les différents états du modèle de Markov caché.

Soit L l'ensemble des comportement possibles, vus comme des étiquettes à ajouter aux observations brutes, et Φ l'espace des observations brutes. On définit un étiquetage partiel qui lie une observation brute $\tau = (o_1, o_2, \ldots, o_n)$ à une observation brute étiquetée $\tau' = ((o_1, l_1), (o_2, l_2), \ldots, (o_n, l_n))$ avec $o_i \in \Phi$ et $l_i \in L \cup \{noMark\}$, et $noMark$ est une étiquette vide utilisée si aucune étiquette n'est donnée par l'expérimentateur.

On demande à l'observateur, lors de la collecte des données brutes pour la construction de modèle d'étiqueter quand il peut (mais le plus souvent possible quand même) par un comportement dans L.

Lors de la phase d'extraction des caractéristiques (section II.2), les étiquettes sont conservées sur les vecteurs de caractéristiques. Si aucune étiquette n'est disponible sur la fenêtre temporelle courante, on donne la valeur $noMark$ pour l'étiquette du vecteur, et si plusieurs sont disponibles (c'est-à-dire que l'observateur a donné plusieurs étiquettes dans un délai inférieur à la largeur de la fenêtre temporelle), on prend la dernière, car on suppose que l'observateur aura toujours tendance à être en retard par rapport à l'état réel du robot (temps de reconnaissance et décision de l'observateur, et temps de saisie de la nouvelle étiquette).

Les vecteurs de caractéristiques sont classifiés comme expliqué dans la section II.2, c'est-à-dire sans tenir compte des étiquettes. Ensuite, *pour chaque étiquette*, on regarde quelles sont les classes d'observation qui apparaissent, et on les place dans l'espace de leurs vecteurs caractérisants comme illustré sur la figure II.7. On définit ensuite un seuil pour la distance entre deux vecteurs caractérisants, de façon arbitraire. Entre chaque paire de vecteurs caractérisants dont la distance est inférieure au seuil, on place alors une arête. On obtient donc un graphe dans l'espace des vecteurs caractérisants dont les sommets sont les vecteurs caractérisants issus de la classification des données traitées, et les arêtes représentent des ressemblances entre ces vecteurs caractérisants. On fait alors dans le graphe ainsi construit une recherche de cliques. Ces cliques représenteront donc des ensembles de vecteurs caractérisants fortement corrélés. Chaque clique sera alors un sous-comportement de l'étiquette courante, et son vecteur caractérisant sera alors le barycentre de la clique. On recommence ensuite cette opération pour les autres étiquettes. L'implémentation utilise un seuil pour la distance variable de façon a avoir à peu près le même nombre de cliques par étiquette.

Cette procédure nous permet de définir des sous-états dans ceux donnés par le concepteur du modèle, mais également de donner un bon modèle a priori. En effet, pour chaque observation issue de la classification, on mesure la distance dans ce même espace entre le vecteur caractérisant de l'observation et toutes les cliques de toutes les étiquettes (voir figure II.8). La probabilité de faire l'observation correspondante en étant dans un état est déduite directement de ces distances comme étant la distance au barycentre de la clique correspondante normalisée. Ce processus original est également détaillé formellement dans [Fox et al., 2006].

Nous allons détailler un exemple d'utilisation de cet algorithme dans le cadre de notre exemple dans la section II.6.

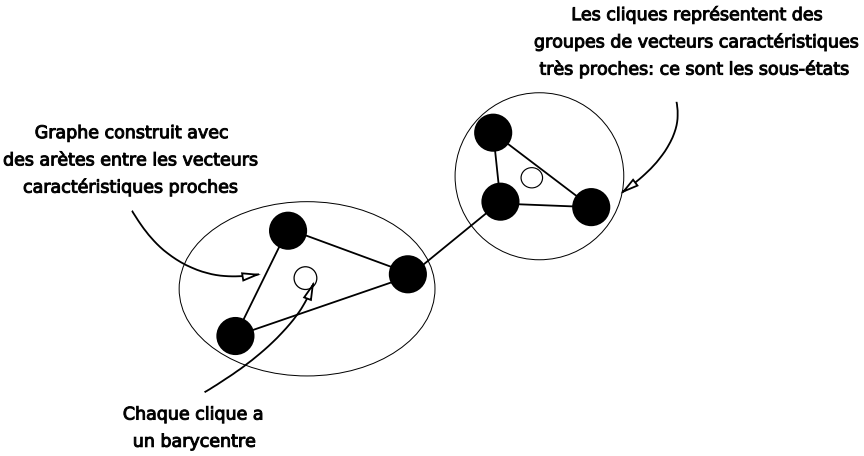


FIG. II.7: Définition des sous-comportements

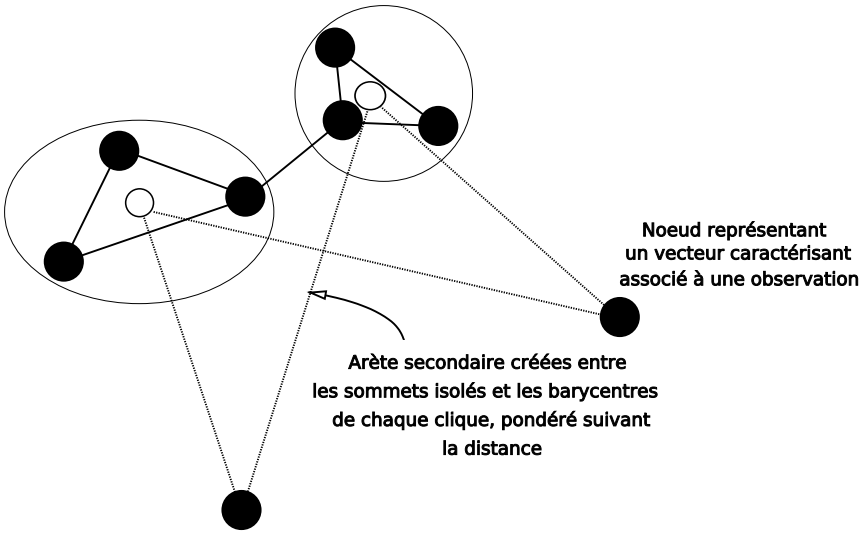


FIG. II.8: Définition du modèle d'observation a priori

II.5 Utilisations

Maintenant que nous avons vu comment construire un modèle de Markov caché, examinons les utilisations possibles de ce type de modèle. Elles se découpent principalement en deux catégories : la reconnaissance de processus, et la prévision au sens large.

Reconnaissance

Une utilisation courante des modèles de Markov caché est la reconnaissance de processus. Cette approche est utilisée avec succès en reconnaissance de la parole [Rabiner, 1989], mais aussi en reconnaissance de gestes devant une caméra [Yoon, Soh, Bae et Yang, 2001; Wilson et

Bobick, 2001; Bobick et Davis, 2001; Wilson et Bobick, 1999; Nam et Wohn, 1996] ainsi que pour la reconnaissance d'expression faciales [Cohen, Sebe, Chen, Garg et Huang, 2003]. Si les principaux apports de ces travaux sont au niveau du choix et de l'extraction des caractéristiques utiles et efficaces (ce qui est un problème très délicat comme nous l'avons déjà souligné), la reconnaissance elle-même se fonde toujours sur le même procédé. Pour cela, le but est de déterminer quel est le processus que l'on est en train d'observer parmi m processus déjà modélisés. Lorsque l'on a appris m différents modèles λ_m pour chaque activité à reconnaître, et que l'on dispose d'un séquence d'observation O dont on ne sait pas à quelle activité A_m elle correspond, on calcule :

$$p_m = P(O|\lambda_m) \tag{II.59}$$

pour chaque modèle. Ceci est fait rapidement grâce à la terminaison du calcul récursif des α_t, comme indiqué dans l'équation II.25. Ce calcul est en général rapide (si la séquence d'observations O n'est pas trop longue).

Le modèle le plus probable est alors λ_m qui maximise $P(O|\lambda_m)$, soit :

$$m = \underset{m}{argmax}\, p_m = \underset{m}{argmax}\, P(O|\lambda_m) \tag{II.60}$$

Si la reconnaissance peut être faite en un temps court lorsque la séquence O est courte elle-même, elle nécessite quand même en théorie toute la séquence d'observation pour calculer les p_m. On peut imaginer utiliser une observation partielle, mais les résultats ne sont alors pas garantis. Nous verrons dans la section IV.1 comment d'autres types de modèles stochastiques peuvent alors se révéler plus appropriés pour ce type de tâche.

Néanmoins, l'extraction des caractéristiques reste un problème majeur pour la reconnaissance rapide. Si certaines caractéristiques sont très complexes à calculer, la reconnaissance en ligne devient simplement impossible.

Suivi

On peut également utiliser un HMM seul, à l'intérieur d'un processus, pour deviner la valeur de l'état en un instant t. Deux cas se présentent alors : soit on dispose de la séquence d'observation totale, et l'on parlera alors de suivi après-coup, soit on veut faire du suivi en ligne, et l'on parlera alors de prédiction.

Dans le premier cas, une solution consiste à utiliser simplement les valeurs de $\gamma_t(i)$ selon l'équation II.17, déduit de la séquence globale par l'équation II.29. On a alors :

$$q_t = \underset{1\leq i\leq N}{argmax}[\gamma_t(i)] \qquad 1 \leq t \leq T \tag{II.61}$$

Cependant, si on déduit l'état le plus probable pour tout t, la séquence d'états correspondante peut être mauvaise : par exemple si une transition est impossible (le a_{ij} correspondant est nul), elle peut quand même apparaître dans la séquence trouvée, car le critère utilisé pour calculer les $\gamma_t(i)$ ne prend pas en compte la séquence d'états. Il faut donc maximiser $P(Q|O,\lambda)$ avec $Q = (q_1, q2, \ldots, q_t)$. Ceci est fait par l'algorithme de Viterbi.

Si $O_{1:T} = (O_1, O_2, \ldots, O_T)$ et $Q = (q1, q2, \ldots, q_T)$, on définit pour tout i la probabilité d'une séquence d'états se terminant en S_i et d'une séquence d'observations :

$$\delta_t(i) = \max_{q1,q2,\ldots,q_{t-1}} P(q_1, q_2, \ldots, q_t = S_i, O_{1:t}|\lambda) \tag{II.62}$$

On peut déduire alors par induction que :

$$\delta_{t+1}(j) = [\max_i \delta_t(i)a_{ij}]b_j(O_{t+1}) \tag{II.63}$$

Il reste alors à mémoriser l'argument qui a maximisé II.63, dans ψ. $\psi_t(j)$ est alors un tableau contenant l'état qui a maximisé II.63, pour tout t et tout j. On obtient alors l'algorithme complet :

1. Initialisation :

$$\delta_1(i) = \pi_i b_i(O_1) \qquad 1 \leq i \leq N \tag{II.64}$$
$$\psi_1(i) = 0 \tag{II.65}$$

2. Induction :

$$\delta_t(j) = \max_{1\leq i\leq N}[\delta_{t-1}(i)a_{ij}]b_j(O_t) \qquad 2 \leq t \leq T \qquad 1 \leq j \leq N \tag{II.66}$$
$$\psi_t(j) = \operatorname*{argmax}_{1\leq i\leq N}[\delta_{t-1}(i)a_{ij}] \qquad 2 \leq t \leq T \qquad 1 \leq j \leq N \tag{II.67}$$

3. Terminaison :

$$p^* = \max_{1\leq i\leq N} \delta_T(i) \tag{II.68}$$
$$q_T^* = \operatorname*{argmax}_{1\leq i\leq N} \delta_T(i) \tag{II.69}$$

4. la séquence est alors (en partant de la fin) :

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad t = T-1, T-2, \ldots, 1 \tag{II.70}$$

De cette façon, on peut obtenir la séquence d'états la plus probable pour une observation complète. On peut alors après-coup caractériser précisément ce qu'il s'est passé lors de l'exécution du processus. Néanmoins, il peut être intéressant de suivre un processus lors même de son exécution, et pas seulement après-coup.

Prévision

Pour faire du suivi en ligne du processus modélisé (si les observations traitées sont disponibles à une fréquence suffisante), on peut s'intéresser à l'état courant, pour deviner où en est le processus à l'instant t. On dispose alors de $O = (O_1, O_2, \ldots, O_t) = O_{1:t}$, on note Q_t l'ensemble sur i des $q_t = S_i$ et on cherche :

$$P(q_t = S_i|O_{1:t}) \tag{II.71}$$

On remarque alors que cette formulation est très proche du calcul des $\alpha_t(i)$ définis dans la section II.3. En effet :

$$\alpha_t(i) = P(O_{1:t}, q_{i;t}) \quad \text{(II.72)}$$
$$\alpha_t(i) = P(q_{i;t}|O_{1:t})P(O_{1:t}) \quad \text{(II.73)}$$

La valeur de $\alpha_t(i)$ peut donc être vue comme un message se propageant en avant constitué de la probabilité jointe d'arriver en un état pondéré par la probabilité de faire l'observation jusque là, et qui nous permet de déduire la valeur courante des probabilités sur la variable d'état.

Ceci peut aussi s'écrire (si on reprend le calcul des α_t, et $P(O_{1:t})$ est un facteur de normalisation) :

$$P(q_{j;t+1}|O_{1:t+1}) = N.P(O_{1:t})P(O_{t+1}|q_{j;t+1}) \sum_{i=1}^{N} P(q_{j;t+1}|q_{i;t})P(q_{i;t}|O_{1:t}) \quad \text{(II.74)}$$

Il est important de noter que le temps et la mémoire requis pour chaque mise à jour est constant dans le cadre des HMM. Ceci nous permet de suivre le processus sur une durée quelconque (encore une fois si les observations sont calculables en ligne). Nous verrons (section III.2) que ce genre d'inférence exacte ne se généralise pas à des modèles stochastiques plus complexes.

Prédire l'évolution du système pour détecter de futurs cas d'échecs est également possible : il suffit de faire le même type d'inférence sans prendre en compte les futures observations. On peut alors inférer (en omettant les probabilités d'observation dans l'équation II.74) :

$$P(q_{j;t+k+1}|O_{1:t}) = \sum_{S_i} P(q_{j;t+k+1}|q_{i;t+k})P(q_{i;t+k}|O_{1:t}) \quad \text{(II.75)}$$

De cette façon, on peut, si l'on suit l'évolution du processus en ligne, également prévoir l'évolution et la probabilité d'atteindre certains états (cachés) dans le futur.

II.6 Application

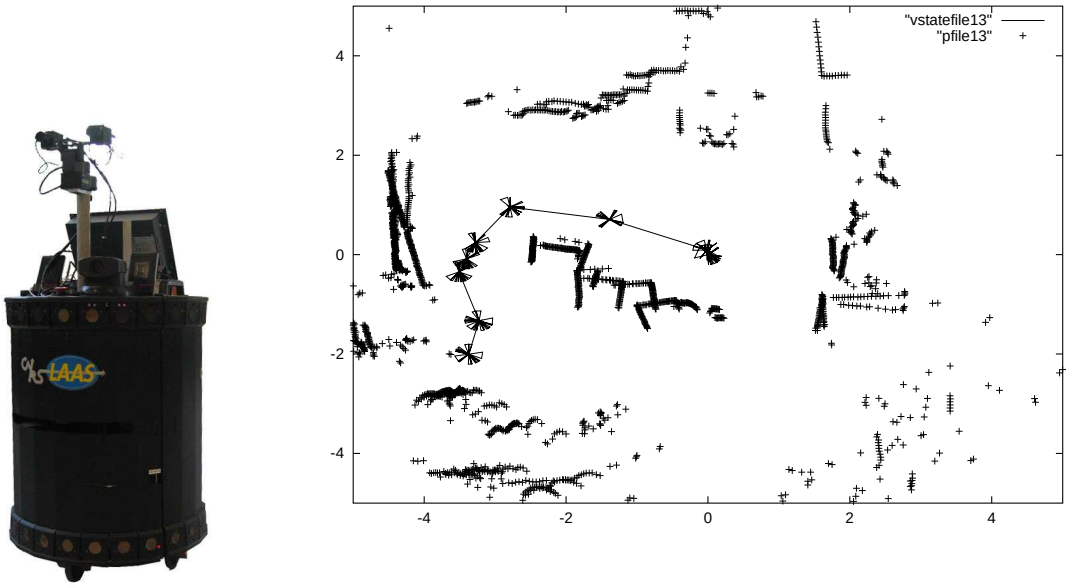
Dans cette section, nous allons présenter dans un premier temps les expérimentations que nous avons effectuées, puis nous parlerons de l'évaluation du modèle de Markov caché appris, et enfin nous détaillerons les avantages du "state-splitting".

Plate-forme expérimentale

Nous avons effectué un grand nombre d'expérimentations sur un nomadic XR4000 visible sur la figure II.9(a). L'architecture logicielle est une architecture originale développée au LAAS [Alami et al., 1998, 2000]. Les fonctionnalités sensori-motrices sont programmées séparément dans des modules logiciels fonctionnels en utilisant l'outil GenOM [Fleury et al., 1997]. L'environnement est perçu via un télémètre laser. La navigation elle-même est basée sur le principe d'évitement réactif d'obstacles ND [Minguez et al., 2004], qui est chargé de la construction

d'une carte locale, de l'évitement d'obstacles et de la génération de consignes en vitesses. La localisation est basée sur l'odométrie uniquement dans nos expérimentations.

Du point de vue du robot, l'environnement se compose donc de points représentants les obstacles, et il construit des vallées traversables qui sont des secteurs angulaires représentés sur la figure II.9(b) par de petits triangles autour de sa position. Les obstacles sont mesurés du point de vue du robot, et comme sur cette figure plusieurs enregistrements successifs sont montrés, on peut nettement deviner la dérive de l'odométrie par le mauvais alignement des obstacles depuis plusieurs points de vues.



(a) Le robot diligent (b) Environnement vu par le robot

FIG. II.9: Plate-forme expérimentale

Cette modalité de fonctionnement est schématisée sur la figure II.10.

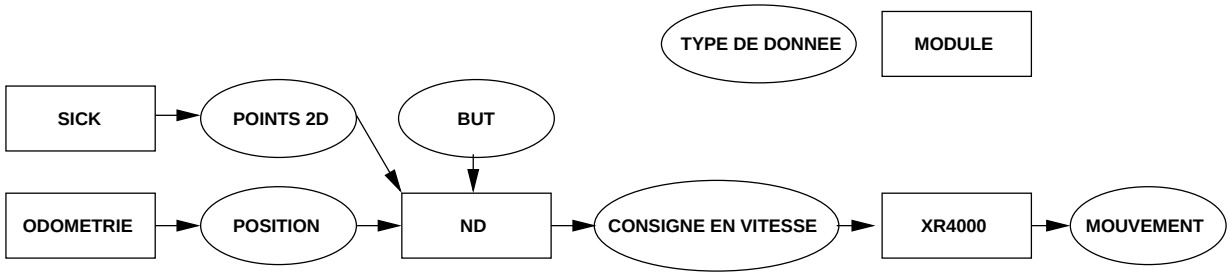


FIG. II.10: Modalité de fonctionnement utilisée

Nous avons enregistré les données brutes sur une soixantaine de navigations, chaque navigation a eu lieu en environnement encombré sur une dizaine de mètres. Les données brutes ont été enregistrées à une fréquence de 5 Hertz, incluant les données brutes détaillées dans la section II.2.

Les données caractéristiques sont alors extraites des données brutes en faisant un traitement sur une fenêtre temporelle de 6 secondes. Cette grande largeur a été choisie après différents essais, elle est due à la faible dynamique du système. En effet, le robot se déplace à une vitesse maximale de 10 cm/s dans des environnements très dégagés. Sa moyenne est autour de quelques centimètres par seconde dans les environnements fortement encombrés qui ont servi pour la collecte de données. Si on prend une fenêtre plus étroite, on a alors énormément de bruit dans les mesures car toutes les valeurs sont très faibles.

Les données caractéristiques sont détaillées dans la section II.2. Leur choix est extrêmement dépendant de ce que va dire l'expert du mode de navigation, ainsi que des comportements macroscopiques que l'on cherche à différencier. Dans notre cas, on utilise la distance parcourue ainsi que celle au but pour différencier les états de *début* et *fin* ; la distance curviligne parcourue sur la fenêtre temporelle ainsi que le changement de cap global doit a priori permettre de dégager les états *progrès* et *recherche*. Le degré d'encombrement doit permettre de différencier l'état d'*évitement*, et finalement la somme des valeurs absolues de changements de cap a été sélectionnée car elle semble très caractéristique de l'état d'*hésitation*.

Les vecteurs de données sélectionnées sont alors classifiées grâce à une carte de Kohonen, comme détaillé plus haut.

Évaluation

Dans notre application robotique, ceci peut nous permettre à la fois un contrôle d'exécution fin pro-actif, et non uniquement de faire de la reprise d'erreur. Les échecs éventuels (états sous-comportements d'*échec*) peuvent être prédits à l'avance. On peut également calculer l'espérance du temps nécessaire à l'accomplissement de la tâche de navigation modélisée.

Ceci peut se généraliser à la gestion de ressources, et de façon générale à tous les aspects d'une tâche de haut niveau. Un tel type de modèle peut donc donner des informations très précises, souvent plus qu'un modèle donnée par un expert de la tâche.

Les résultats ont été obtenus particulièrement par D. Long et M. Fox et sont longuement détaillés dans [Fox et al., 2006], en particulier par l'utilisation d'un facteur de confusion permettant de mesurer plus finement les bonnes prédictions par rapport à des prédictions au hasard. Nous avons comparé les étiquettes données par une explication a posteriori (algorithme de Viterbi) suivant les observations. Nous avons ainsi évalué le score d'un tel modèle de Markov qui est le pourcentage de bonnes étiquettes par rapport aux étiquettes données par l'observateur. On peut voir sur la figure II.11 en abscisse le score obtenu et en ordonnée le nombre d'évaluations qui ont donné ce score. Ainsi si 3 évaluations ont obtenu des scores de 60% de bonnes réponses, on place un point en (60, 3). Une bonne courbe est alors une courbe élevée (beaucoup d'évaluations) sur la droite (ayant un bon score).

Le nombre total d'évaluation est 58, et le score moyen est de 76,18%.

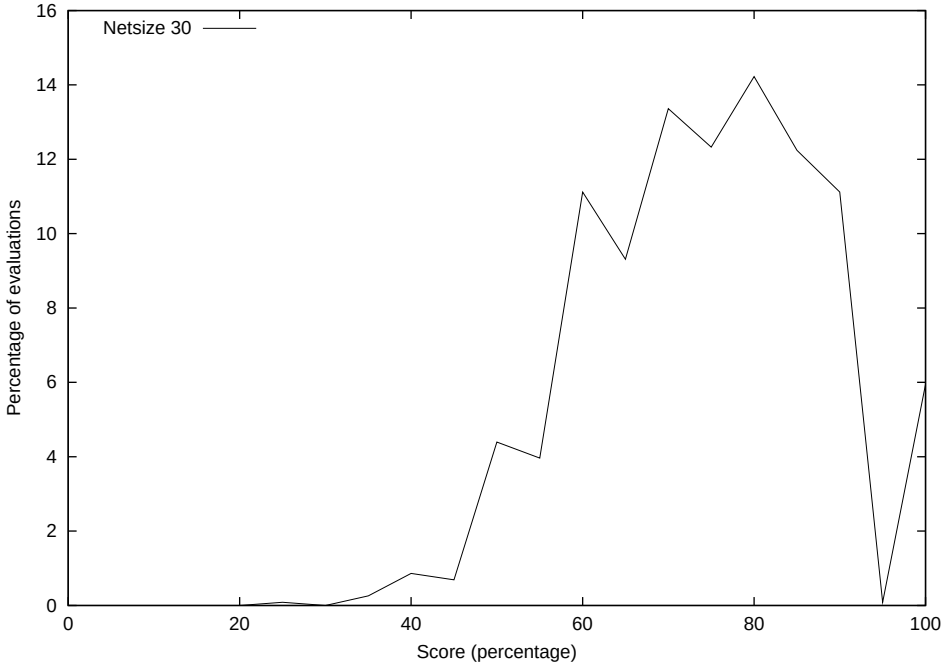


FIG. II.11: Comparaison des étiquettes obtenues par une séquence de Viterbi.

Découpage des états

Dans cette application, on construit 19 observations différentes, et on dispose de 7 étiquettes, incluant l'étiquette vide. On peut voir sur le tableau II.1 le nombre d'occurrences de chaque étiquette pour chaque classe d'observation.

Ensuite, pour chaque marque, on sélectionne les classes qui apparaissent le plus souvent, de façon à limiter le bruit dû à l'imprécision du marquage. Les classes retenues sont marquées en gras, les autres sont simplement ignorées. Ensuite, par colonne, on fait une recherche de clique dans l'espace des coordonnées des vecteurs caractéristiques des observations (figurés sur le tableau II.2).

Par exemple, pour l'étiquette "début", les distances euclidiennes sont sur le tableau II.3.

On place un lien d'une part entre les observations 4 et 15 et un autre entre les observations 15 et 17. En revanche, 4 et 17 sont trop éloignés par rapport au seuil fixé (dans notre implémentation, le seuil est variable de façon à garder à peu près le même nombre de cliques par marque, ici 2 ou 3). Dans ce cas, une recherche de clique dans un tel graphe donnera d'une part la clique (4,15) et d'autre part (15,17). En itérant pour chaque étiquette, on obtient alors les cliques figurées sur le tableau II.4.

A partir de là, on considère simplement que l'on a deux états cachés correspondant à un début, deux également pour une hésitation...

Pour la construction du modèle initial des capteurs, i.e. $P(Observation|Etat)$, on vient simplement redistribuer les probabilités suivant la distance au centre de chaque clique. Par exemple, pour l'observation 13, qui apparaît à la fois dans fin $c_1 = (5, 13)$ et dans progrès $c_2 = (0, 1, 5, 7, 13, 15)$,

classe	début	hésitation	évitement	fin	progrès	recherche	aucune
0		22	104	52	**363**	**385**	**141**
1		**128**	**248**	19	**587**	76	**125**
2	10	39	**242**		147	49	76
3		6	143		216	38	59
4	**34**	**264**	**480**		120	**370**	**174**
5		39	59	**244**	**728**	123	80
6	6	3	47		238	28	20
7		36	122		**288**	89	104
8	3	45	**249**		92	**215**	71
9	31	13	31		63	21	42
10	18	1	126		141	93	6
11			38		180	12	26
12	16	**904**	**435**	43	184	**435**	**607**
13			10	**282**	**453**	20	51
14	1	42	28		39	78	54
15	**91**	14	25		334	33	55
16	27	14	21		**188**		60
17	**135**	5			174	1	48
18		**351**	**493**	27	180	**606**	**313**

TAB. II.1: Comptage des étiquettes par observation

classe	dist. début	dist. curvi	δ cap	ang-diff	encomb.	dist but	vitesse	accélération
0	0.421561	0.000932956	0.272209	-0.46719	0.499863	0.232686	-0.0270574	0.474534
1	-0.17238	0.416905	-0.370812	-0.399366	-0.175066	0.378728	0.336691	0.460474
2	-0.453472	0.106286	0.0150332	-0.433615	-0.160166	0.583052	0.0629518	0.474608
3	-0.399838	0.364502	-0.249746	-0.416931	-0.125174	0.531564	0.31806	0.267717
4	**-0.182129**	**-0.209636**	**0.315388**	**-0.461909**	**-0.286566**	**0.506573**	**-0.239743**	**0.462429**
5	0.374745	0.354845	-0.0980796	-0.460267	0.558326	0.183714	0.37638	-0.158257
6	-0.370585	0.491642	-0.29952	-0.342609	0.186864	0.428047	0.432876	0.0913114
7	-0.157338	0.314217	-0.486115	-0.497301	-0.226675	0.473269	0.342693	-0.00959986
8	-0.518525	-0.00889749	0.0634586	-0.465426	-0.202174	0.670198	-0.0204055	0.141237
9	-0.472537	0.31579	-0.256391	-0.424557	-0.0774065	0.553315	0.330247	-0.0989418
10	-0.575822	0.137223	-0.0365915	-0.454145	-0.0381672	0.648753	0.137437	0.0280462
11	-0.184529	0.523988	-0.375208	-0.381181	0.0564246	0.364975	0.510575	-0.090706
12	0.131442	-0.358877	-0.278191	-0.403798	-0.390035	0.311577	-0.367733	-0.478593
13	0.280216	0.472549	-0.318185	-0.289754	0.371004	0.0587546	0.47798	-0.378703
14	-0.187086	0.0897807	0.0509308	-0.64766	0.0324369	0.67746	0.109819	-0.250634
15	**-0.445248**	**0.377653**	**-0.133589**	**-0.322076**	**0.403066**	**0.482558**	**0.370701**	**-0.069352**
16	-0.429483	0.387011	-0.289805	-0.241049	0.349198	0.386117	0.378885	-0.330298
17	**-0.453175**	**0.363576**	**-0.250907**	**-0.222409**	**0.43496**	**0.391589**	**0.363659**	**-0.274308**
18	0.309233	-0.375967	0.541248	-0.364316	-0.128322	0.266881	-0.382041	0.322026

TAB. II.2: Coordonnées des classes d'observations

on calcule les barycentres de chacune de ces cliques (b_1 et b_2), et alors :

$$P(13|c_1) = \frac{d(13, b_1)}{d(13, b_1) + d(13, b_2)}$$

observations	distance
4,15	1.33
4,17	1.48
15,17	0.87

TAB. II.3: Distances entre les observations pour la marque "début"

Étiquette	clique	clique	clique
Début	4 15	15 17	
Hésitation	1 4	4 12 18	
Évitement	1 2 4 8	2 4 8 12 18	
fin	5 13		
progrès	0 1 5 7 13 15		
recherche	0 4 8 18	4 8 12 18	
aucune	0 1 4	0 4 18	4 12 18

TAB. II.4: Cliques contruites par étiquette

Pour évaluer cet algorithme, nous avons effectué le même test que précédemment. On peut voir sur la figure II.12 les deux courbes correspondantes. On voit que les scores réalisés sans découpage d'états sont nettement pires.

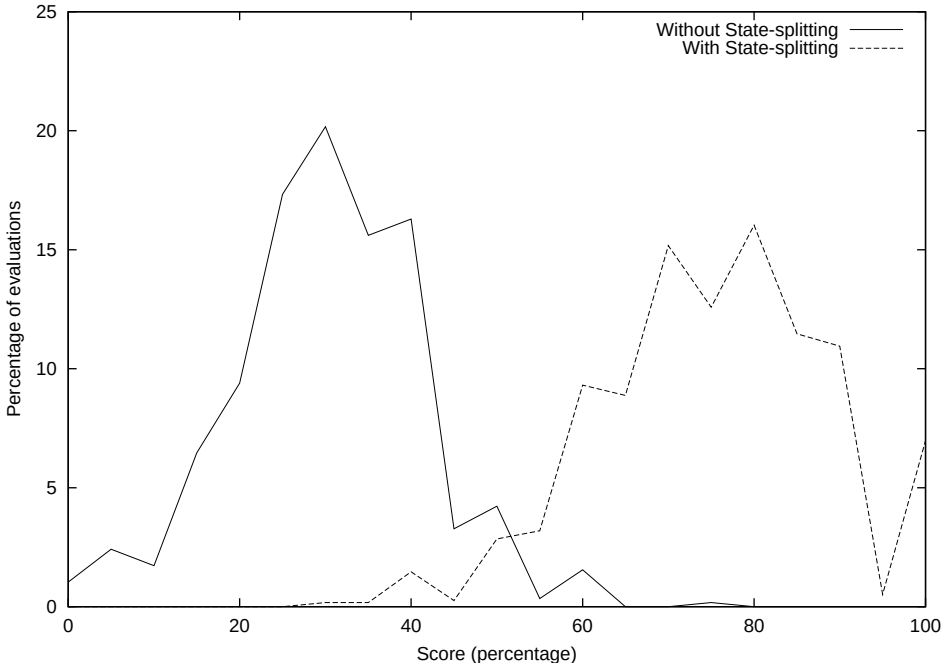


FIG. II.12: Scores obtenus avec et sans découpage d'états.

Finalement, nous illustrons l'interêt d'un bon point de départ pour les fonctions d'observa-

tions sur la figure II.13. On voit bien qu'avec une distribution aléatoire (notée RSM) le modèle appris est de très mauvaise qualité par rapport au modèle appris avec un modèle des observations obtenu comme décrit dans la section II.4.

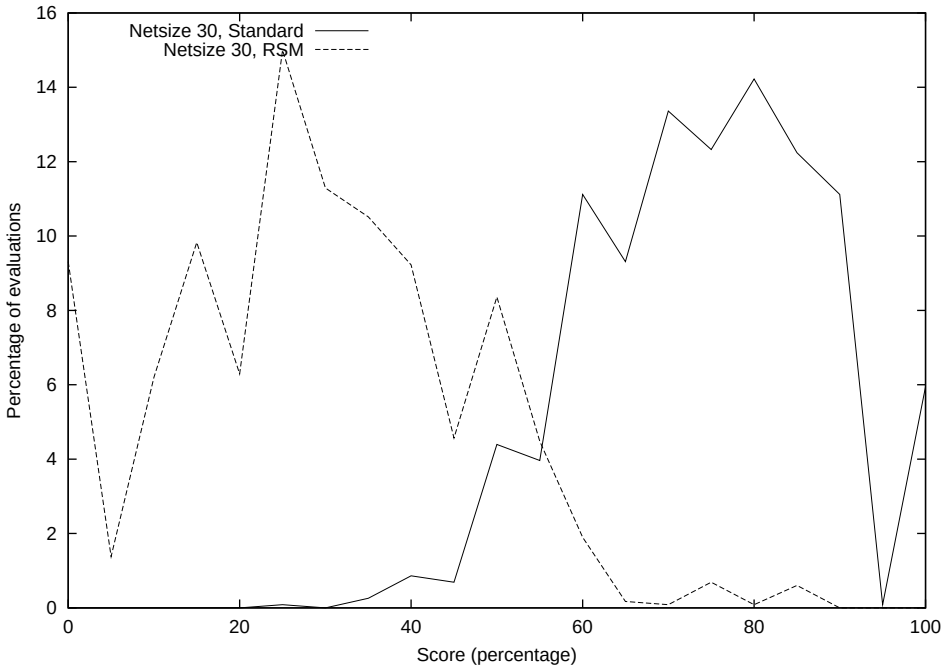


FIG. II.13: Scores obtenus avec modèle des observations aléatoire (RSM) et avec un modèle des observations obtenu par redistribution grâce au découpage d'états.

II.7 Discussion

Plusieurs difficultés sont fondamentales lors de l'apprentissage de modèles à variables cachées.

- *Le nombre de valeurs possible pour l'état caché (le nombre d'états cachés)*. Dans certaines applications comme la reconnaissance de parole, on peut prendre le nombre de phonèmes. Mais pour modéliser un système plus complexe, on ne sait pas quel est le nombre idéal. Nous avons proposé un algorithme qui mixe connaissance subjective humaine et données capteurs et qui donne de bons résultats. On n'a bien sûr aucune garantie de tomber sur le bon nombre d'états, mais cette approche semble être un bon compromis et permet de plus de conserver une certaine sémantique aux états cachés, qui ne sont alors plus des variables muettes. De plus les résultats ne vont pas changer dramatiquement avec un état de plus ou de moins. Ce qui est important, c'est d'avoir une idée de l'ordre de grandeur du nombre d'états avant apprentissage.
- *Le modèle initial avant apprentissage*. Il ne faut surtout pas oublier que l'algorithme EM est une recherche locale, et que l'espace des modèles stochastiques a forcément une forme

très complexe. Le point de départ de l'apprentissage est alors extrêmement sensible. Encore une fois, si le nombre d'états cachés a été choisi au hasard, il va forcément en être de même pour le modèle initial. Et dans ce cas, l'apprentissage du modèle risque fortement de s'arrêter sur un optimum local. La méthode que nous avons proposé permet d'avoir au moins un modèle des capteurs (transitions état vers observations) particulièrement bon, en redistribuant les probabilités d'observation de telle ou telle classe sur les états cachés, car ceux-ci sont définis aussi d'après les observations.

– Un point dernier point crucial est *le pré-traitement des données*, c'est-à-dire la sélection des caractéristiques. La quantité de données brutes et raffinées disponibles peut être absolument énorme (limitée par l'imagination du concepteur), et choisir les fonctions de traitement correctes de façon à extraire des données utiles pour modéliser le système est un problème ouvert. Néanmoins, le principe de la fenêtre glissante pour lisser les données et pour contourner l'hypothèse markovienne est très général. De la même façon, utiliser une classification non supervisée est intéressant pour éliminer un tel biais. Si de telles techniques d'apprentissage statistique sont indispensables, aucune n'est malheureusement applicable sur les données brutes (comme par exemple le nuage de point renvoyé par le télémètre). La qualité d'un modèle se joue souvent sur cet aspect.

Chapitre III

Modèles Stochastiques Causaux

Dans ce chapitre, nous allons nous servir d'une généralisation des modèles de Markov cachés, dans laquelle la structure causale peut être entièrement explicitée. Ce type de modèle garde des HMMs la représentation temporellement repliée, et l'on n'explicite que la causalité entre deux pas de temps successifs, cette structure restant statique au cours du temps.

L'utilisation de représentation plus structurée que le cadre HMM dans le cadre de notre application à la navigation robotique a pour but de mettre en évidence les effets des paramètres contrôlables du robot. Nous avons en effet en vue de pouvoir utiliser une modélisation pour optimiser finement le comportement du robot, et ceci n'est pas directement faisable en utilisant des HMMs. Nous verrons au chapitre V comment utiliser les modèles complexes étudiés dans ce chapitre pour cela.

Nous allons dans un premier temps présenter formellement ce type de modèles, puis nous allons expliciter les différences fondamentales que cette représentation entraîne par rapport au cadre HMM. Nous montrerons ensuite une formalisation originale de la manière d'apprendre quantitativement un tel modèle, qui constitue la contribution principale de cette partie. Puis nous parlerons de l'apprentissage de la structure causale elle-même. Nous évoquerons ensuite des pistes pour l'apprentissage incrémental d'un tel modèle. Enfin nous montrerons une application que nous avons réalisée.

III.1 Réseau Bayésien Dynamique

Un réseau bayésien dynamique (DBN pour Dynamic Bayesian Network) ou réseau probabiliste dynamique [Dean et Kanazawa, 1990] est constitué d'un ensemble de variables cachées, de variables observables, ainsi que d'une structure causale entre ces variables (c'est-à-dire un ensemble de liens causaux). C'est un réseau bayésien dans lequel les variables sont indicées par le temps, et donc chaque variable a une instance à chaque pas de temps, et dans lequel la structure causale se répète indéfiniment, c'est-à-dire que les liens causaux sont les mêmes quelque soit le pas de temps.

On peut voir les DBN comme une généralisation des modèles de Markov cachés, dans lequel on n'aurait plus une seule variable cachée et une observable liées par la structure très simple dé-

taillé dans la section II.1, mais N variables cachées et M variables observables, avec quasiment n'importe quel type de structure causale entre elles. La seule contrainte est que les liens causaux cycliques sont interdits, comme dans un réseau bayésien (et sont généralement dépourvus de sens), et que l'hypothèse de Markov reste applicable, c'est-à-dire que les liens causaux influençant une variable à l'instant t ne peuvent provenir que de variables à l'instant t ou à l'instant $t-1$. Ceci nous permet d'avoir une représentation très compacte et visuelle d'un réseau baysien dynamique, puisqu'il suffit de représenter les variables aux instants t et $t-1$ avec leurs liens causaux pour visualiser tous les liens causaux de la structure. Finalement, comme pour les modèles de Markov cachés, les liens causaux sont constants tout au long du temps autant en structure qu'en quantité (ils sont probabilistes, et les probabilités sont indépendantes du temps).

Définition

Un DBN est un modèle discret de transitions stochastiques entre un certain nombre de variables aléatoires, et qui décrit une étape de la dynamique de transition du processus. L'hypothèse d'invariance temporelle nous assure que ce modèle représente la dynamique du processus à n'importe quel instant. Le modèle de transition est décrit comme un $k-$TBN, c'est-à-dire un réseau bayésien temporel à k tranches. Un $k-$TBN est un fragment de réseau bayésien sur un ensemble de variables corrélées temporellement, et qui représente une *transition stochastique semi-Markovienne d'ordre* $k-1$, c'est-à-dire les distributions de probabilités conditionnelles sur les variables étant données les $k-1$ valeurs précédentes.

Le terme réseau Bayésien dynamique est utilisé pour parler d'un 2-TBN, qui est forcément Markovien.

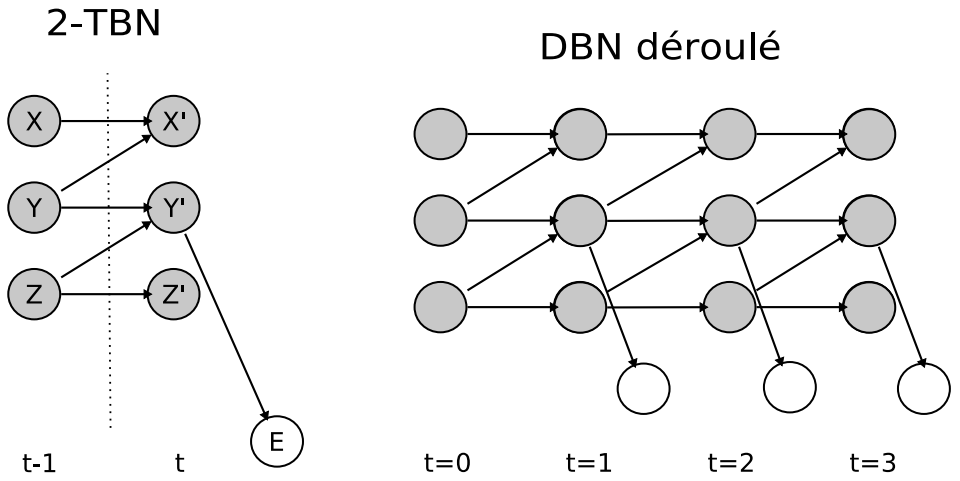
Un réseau bayésien temporel à k tranches temporelles (un $k-$TBN) défini sur un ensemble de variables $U = \{U_1, \ldots, U_n\}$ est un couple $B = (\mathcal{G}, \Theta)$, constitué des composants suivants :

1. un *graphe direct acyclique* (DAG) $\mathcal{G} = (\mathbf{S}, \mathbf{A})$, dont les sommets ($\in \mathbf{S}$) sont organisés en k tranches. Les tranches sont repérées par leur indice temporel relatif dans $\{-k+1, \ldots, -1, 0\}$ ou de façon équivalente par leur indice temporel absolu dans $\{t-k+1, \ldots, t-1, t\}$, avec t la référence temporelle spécifiée*. $\mathbf{S}$ est tel que pour chaque variable $u \in U$ corresponde un et un seul sommet $u^{(t)}$ dans la tranche d'index t et au plus un sommet $u^{(t-d)}$ dans chaque tranche d'index $t-d$ avec $0 < d < k$. $\mathbf{A}$ définit une relation binaire acyclique dans laquelle toutes les arêtes doivent appartenir à $\mathbf{S} \times \mathbf{S}^{(t)}$, et $\mathbf{S}^{(t)}$ est la dernière tranche de V d'index t†.
2. une paramétrisation ou quantification Θ, qui associe à chaque sommet $u^{(t)}$ de la dernière couche de $\mathbf{S}$ une distribution conditionnelle de probabilités $P\left[u^{(t)} | Pa(u^{(t)})\right]$ (et $Pa(u^{(t)})$ sont les parents de $u^{(t)}$ par $\mathbf{A}$).

La sémantique des DBNs est facilement compréhensible si l'on déroule un $k-$TBN (tel que présenté sur la figure III.1(a)) en réseau bayésien ordinaire. Comme on peut le voir sur la figure III.1(b), on peut transformer un $k-$TBN en réseau bayésien simplement en dupliquant les noeuds de la dernière colonne au long du temps, ainsi que leurs probabilités conditionnelles.

*Une variable apparaît forcément une seule fois dans la dernière tranche et au plus une fois partout ailleurs

†ou encore : les arêtes doivent forcément aboutir dans la dernière tranche



(a) Exemple de DBN (les variables grisées représentent des variables cachées)

(b) Le même DBN, déroulé sur 4 pas de temps

FIG. III.1: Un exemple de Réseau Bayésien dynamique

Pour finir de passer à un réseau bayésien, il faut définir une distribution a priori sur les $k-1$ premières tranches du DBN déroulé de façon à avoir une distribution bien définie (car les probabilités conditionnelles dépendent des $k-1$ tranches précédentes). Pour un DBN, on donnera simplement une distribution a priori sur les variables non déductibles (sans lien causal entrant) en $t = 0$. Dans un cadre DBN, on notera N le nombre de variables (cachées ou non). Une

variable est notée q^i avec $0 < i \leq N$. On note M_i l'arité de q^i et q^i peut prendre les valeurs S^i_j avec $0 < j \leq M_i$. On note q^i_t la valeur de la variable q^i à l'instant t. On note λ l'ensemble du DBN. Nous notons O_t la variable composite de toutes les variables observables à l'instant t. Une séquence d'observations sera notée $O_i, \ldots, O_j$ ou plus simplement $O_{i:j}$ avec $i < j$. Chaque parent au sens causal d'une variable est soit dans le pas de temps de la variable fille, soit dans le pas de temps précédent, car on est dans le cas $2 - TBN$. On note l'ensemble des instanciations des parents d'une variable i : $Pa(i)$. On note une instance σ_i telle que le premier parent i^0 de q^i prenne sa ${j_0}^{\text{ième}}$ valeur, le deuxième parent i^1 prenne sa ${j_1}^{\text{ième}}$ valeur... :

$$\sigma_i \in Pa(i) = ({i^0}_{j_0}, {i^1}_{j_1} ... {i^k}_{j_k}) \tag{III.1}$$

Une probabilité de transition d'un ensemble de valeurs des variables parents vers une valeur de la variable fille q^i sera alors :

$$a(\sigma_i, i_j) = P(q^{Pa(i)} = \sigma_i, q^i = S^i_j) \tag{III.2}$$

avec $\sigma_i \in Pa(i)$, et $q^{Pa(i)}$ les parents de la variable i.

Comparaison aux modèles de Markov cachés

Ce type de modèle peut être comparé aux modèles de Markov cachés. Reprenons un exemple robotique pour détailler cela. Précédemment, nous avons défini un HMM à un état caché présentant l'état interne du robot parmi les classes *début, fin, échec, hésitation, progrès* .... Nous avons ensuite défini des sous-classes (voir section II.4), et les états sont donc dans l'espace $classe \times sous - classes$, et les sous-classes sont juste des numéros (variable muette). L'observation était alors un agrégat de différentes variables mesurées et traitées, avec en particulier *distance depuis le début (dd), distance curviligne sur la fenêtre temporelle (dc), changement de cap (cc)* ... (nous ne reprenons pas toutes les variables pour alléger l'argumentaire). L'observation du HMM est donc dans cet espace $dd \times dc \times cc$. Le HMM (en version simplifiée) est donc celui de la figure III.2.

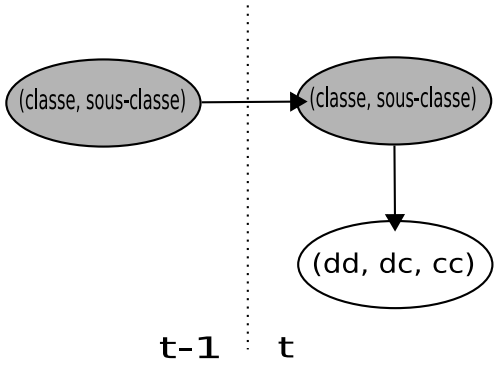


FIG. III.2: Version simplifiée du modèle de Markov caché défini précédemment

Comme nous sommes dans le cadre des modèles de réseau bayésien dynamique, la structure est à définir, en particulier en terme de nombre de variables. Nous pouvons donc très bien ne pas agréger classes et sous-classes, ni non plus les différentes observations en une seule. Cette façon de voir les choses correspond à une *factorisation* de l'espace défini pour les HMMs. En effet, chaque variable définie pour le HMM est représentée ici par plusieurs variables qui sont ses différentes "coordonnées", i.e. ses différentes composantes, séparées en ses différents facteurs. On peut donc construire un DBN explicitant la causalité d'un HMM juste en redécoupant les états du HMM suivant les facteurs que l'on a été obligé d'agréger pour se plier à la structure fixe. Dans un HMM, on est obligé de représenter en extension l'espace $classe \times sous - classe$, alors qu'il est représenté en intention dans cette représentation factorisée. Un état au sens classique du terme est alors représenté par une instanciation de l'ensemble des différentes variables, des différents facteurs qui constituent l'état. On passe donc d'un état global à un ensemble de variables qui peuvent être de sémantiques différentes, comme dans l'exemple classe et sous-classe. De la même façon, on passe d'un système de transition d'états à un système de liens causaux précis plus fins entre chaque composante.

On obtient alors le DBN de la figure III.3, qui est *strictement équivalent*, en ayant bien sûr les quantification adéquates. En effet, chaque variable est déductible des mêmes composantes que précédemment, et les probabilités jointes sont respectées.

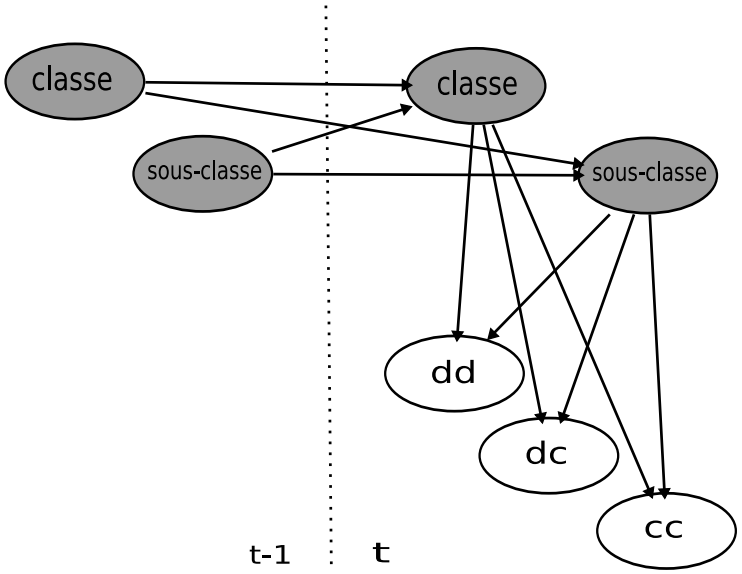


FIG. III.3: DBN correspondant au HMM de la figure III.2

Le lien entre HMM et DBN a été clairement explicité dans [Smyth, Heckerman et Jordan, 1997]. Sans rentrer dans le détail, il est clair qu'un HMM est un cas particulier de DBN, et mieux encore, le cadre DBN nous permet de ne pas être obligé d'agréger des variables naturellement séparées (cas de l'observation). En dehors d'une inclusion théorique des HMM dans le cadre des DBN, cette deuxième approche a deux intérêts majeurs. Le premier est illustré sur la figure III.4 : c'est le cas où certaines observations ne dépendent par exemple que de la classe, et pas de la sous-classe.

Sur la figure III.3 qui est équivalente au HMM III.2, on constate qu'il y a une complétude des liens causaux, c'est-à-dire qu'il n'y a aucune indépendance entre les variables. Même si on a l'indépendance d'une des observations à la sous-classe, on est obligé d'exprimer les probabilité conditionnelles correspondantes. Ce n'est plus le cas pour le DBN III.4, où l'indépendance est exprimée par le retrait du lien causal, ce qui nous permet de rendre la représentation plus compacte. Et les algorithmes utilisés pour les HMMs pourront être optimisés pour utiliser cette compacité, ce qui nous permet d'attaquer des problèmes avec un très grand nombre de variables. En effet, cette représentation factorisée nous permet d'avoir des quantifications de liens causaux de taille raisonnable, ce qui n'est pas le cas pour des HMM avec des variables de très grande arité. Nous y reviendrons.

Le deuxième intérêt concerne une grande limitation des HMMs : l'état interne du système est représenté par une (seule) variable cachée. Or ceci n'est pas forcément réaliste pour un système robotique. En effet, pour la tâche de navigation, l'environnement influence le processus, et il peut être observable, tout comme des paramètres du processus lui-même. Sur la figure III.5, on introduit deux variables e et p, où e est une mesure de l'environnement (par exemple l'encombrement tel que calculé dans la section II.2), et p un paramètre de l'évitement réactif d'obstacles utilisé, par exemple la vitesse linéaire maximale autorisée.

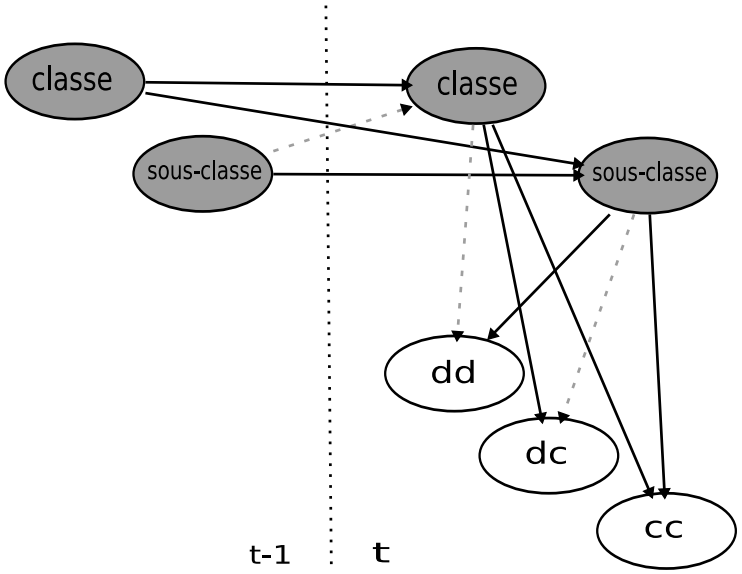


FIG. III.4: Indépendance non modélisable dans le cadre HMM

Dans le cadre HMM (figure III.6), on ne peut pas représenter ces variables simplement, il faudrait intégrer ces variables dans l'état caché, et donc se priver des observations correspondantes, nous avons montré au chapitre précédent comment prendre en compte de telles composantes dans l'état caché, et ces techniques ne permettent pas de rendre explicite des paramètres de contrôle, par exemple. Dans le cadre DBN, toutes les composantes peuvent être explicitées et prises en compte de la même façon.

Il est clair que le cadre DBN est plus large que celui des HMMs. Le pouvoir expressif est plus élevé, et les algorithmes bien adaptés permettent de gagner également en complexité, du fait de l'utilisation de cette représentation factorisée.

III.2 Inférence, à quel coût ?

Regardons d'un peu plus près ce que nous permet de faire une représentation factorisée, et en particulier si la connaissance plus fine de la structure du processus nous permet plus de déductions.

Inférence Exacte

Comme nous l'avons déjà dit, la représentation explicite de tous les liens causaux nous permet de faire apparaître des indépendances, ce qui n'est pas possible avec les HMMs. Malheureusement, une représentation factorisée en de multiples variables, comme c'est possible de le faire avec des DBNs, pose un autre problème fondamental, qui va rendre l'inférence exacte trop complexe.

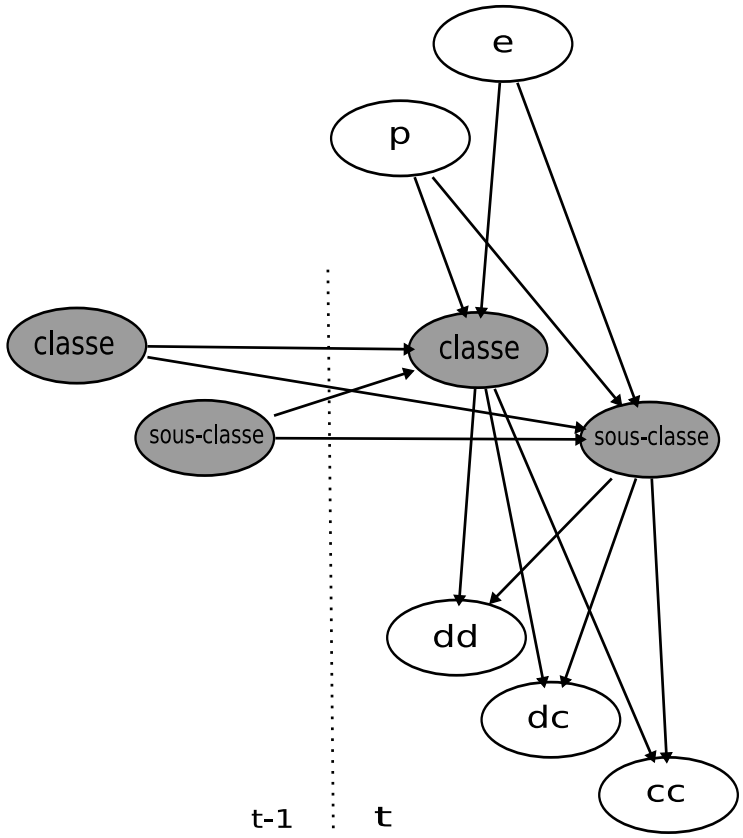


FIG. III.5: Variables observables supplémentaires

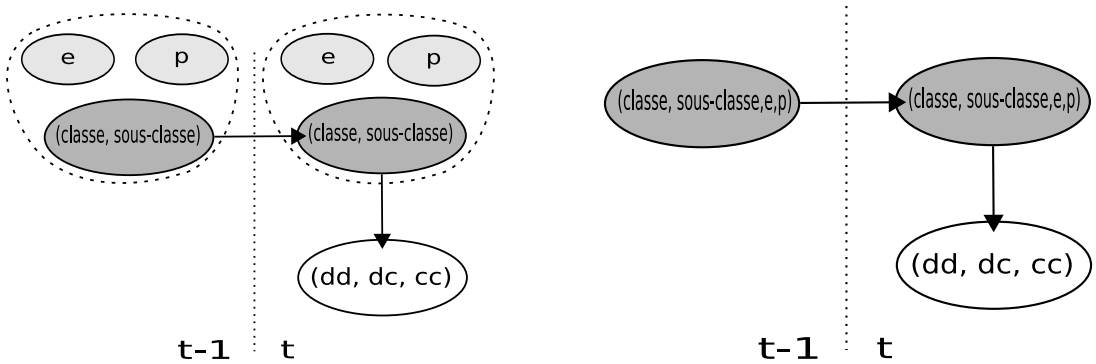


FIG. III.6: Représentation des variables supplémentaires dans le cadre HMM : on est obligé de les cacher et donc de s'en priver lors de l'utilisation du HMM

Le problème est que même si les interactions entre les variables sont très structurées et locales, pour un DBN « creux »(loin d'être complet), les corrélations se propagent rapidement à travers tout le modèle, et en général, toutes les variables sont corrélées à toutes les autres. Dans ce cas, si l'on appelle *état courant cru* une distribution de probabilités sur les variables en un

instant t, alors cet état courant cru n'est pas représentable de façon factorisée.

En effet, il doit contenir toutes les corrélations entre variables pour donner une image exacte du processus. Or suivant les structures, ces corrélations ne sont pas Markoviennes d'ordre 1. Pour tenir compte des corrélations entre un ensemble de variables v à l'instant t, il faut se souvenir des variables qui influencent cet ensemble s à l'instant $t-1$, appelons cet ensemble v'_{t-1}. Malheureusement, il est très rare que les variables de v'_{t-1} ne soient pas elles-mêmes corrélées suivant des dépendances dues à un ensemble $v^2_{t_2}$, etc. Seules certaines structures très simples permettent de n'avoir à maintenir qu'un ensemble fini de corrélations, c'est-à-dire un état courant cru de taille raisonnable.

Comme l'inférence est (par définition) la propagation sur le temps de cet état courant cru (dont la taille augmente exponentiellement avec t), alors l'inférence devient impossible à réaliser.

Pour illustrer cela, considérons le DBN de la figure III.7(a), dans lequel on a beaucoup d'indépendances. Si l'on considère ce même DBN déroulé (figure III.7(b)), on peut voir que toutes les variables sont structurellement corrélées dès l'instant 3. En particulier, toute paire de variable à l'instant 3 a un ancêtre commun au temps 0, dont l'influence se propage via les pas de temps 1 et 2. La figure III.7(b) montre (lien épais) les corrélations nécessaires à garder dans l'état courant cru pour pouvoir quantifier la corrélation entre W^3 et Z^3.

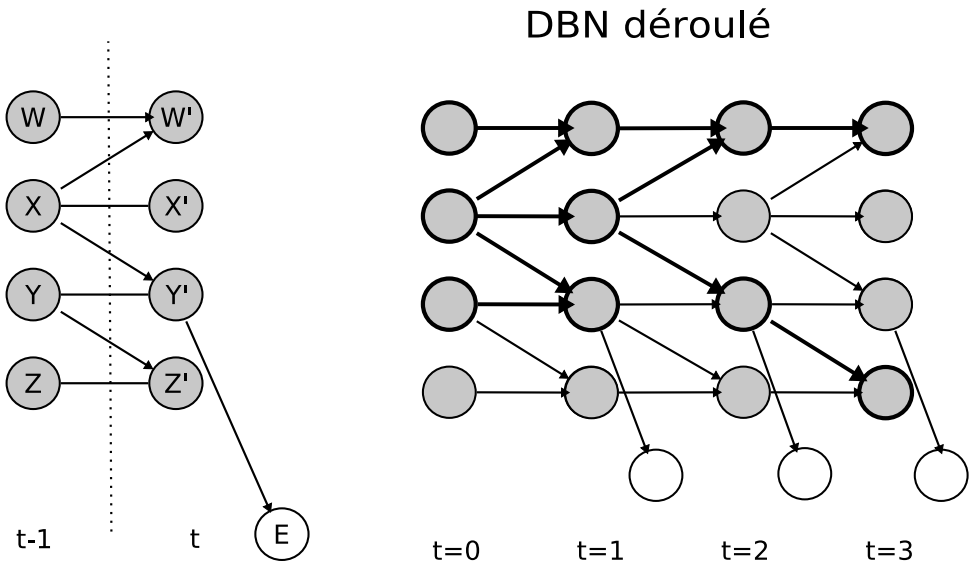


FIG. III.7: Représentation de l'état courant cru nécessaire pour connaître la corrélation exacte entre variables à l'instant 3.

De façon générale, si le processus à modéliser ne se décompose pas en sous-processus bien distincts, alors l'état courant cru ne peut être représenté de façon exacte sous forme factorisée, du fait des corrélations entre variables.

Inférence Approchée

Il est très important pour de nombreuses applications (dont l'apprentissage de modèle) d'avoir une bonne approximation des corrélations entre variables et de leurs valeurs possibles, ceci au niveau quantitatif probabiliste ; mais surtout il est indispensable d'avoir une représentation compacte et pratique d'utilisation. En effet, tous les algorithmes devant traiter avec de tels modèles ont besoin d'une telle représentation.

Différentes approches ont été proposées pour approximer l'état courant cru de façon à en avoir une représentation factorisée, dont la taille n'augmente pas exponentiellement avec t. Certaines visent à décomposer l'état courant cru en clusters de variables faiblement corrélés, et à approximer la corrélation entre les différents clusters [Boyen, 2002; Boyen et Koller, 1998, 1999]. Ces approches sont donc fortement dépendantes du processus à modéliser. D'autres approches ont été proposées [Koller et Fratkina, 1998], dans lequel l'état courant cru est approximé de façon moins formelle, mais aussi de façon plus générale que précédemment.

Une des idées les plus générales est d'estimer l'état courant cru sous forme de filtre particulaire. Un filtre particulaire est composé d'un grand nombre de particules, chacune étant une hypothèse sur l'état courant cru. Chaque hypothèse (chaque particule) est ré-évaluée au fur et à mesure de l'avancement du processus, et garde les corrélations intrinsèques à chaque hypothèse. Ce principe est très général et a été utilisé en statistiques sous le nom de « sequential importance-sampling resampling »(SIR) [Rubin, 1988; Liu et Chen, 1998], en théorie du contrôle sous le nom de filtre particulaire [Gordon, 1994; Gordon, Salmond et Smith, 1993], en intelligence artificielle sous le nom de « survival of the fittest »[Kanazawa, Koller et Russel, 1995] et en vision sous le nom de « condensation »[Isard et Blake, 1996].

Le principe du filtrage particulaire est illustré sur la figure III.8 ; il s'appuie sur le DBN de la figure III.2. Pour la simplicité de la compréhension, on considère que les variables cachées *classe* et *sous-classe* peuvent prendre chacune juste deux valeurs c_0, c_1 et s_0, s_1, respectivement. On peut imaginer par exemple que c_0 signifie comportement correct, c_1 échec, (c_0, s_0) pas de progrès vers le but, (c_0, s_1) avancée vers le but, (c_1, s_0) échec car le robot est bloqué et n'a pas avancé depuis un certain laps de temps et enfin (c_1, s_1) collision avec un obstacle.

Sur ce schéma, une particule est représentée par un rond et est constituée d'une affectation complète des différentes variables *cachées*. Une particule dans la case en haut à gauche signifie qu'elle donne la valeur c_0 à la variable C (de classe) et s_0 également à la variable S (de sous-classe). Le filtre lui-même est l'ensemble des particules. On peut déduire statistiquement de la figure qu'avant propagation on a $\frac{2}{7}$ de chance que les variables C et S aient la valeur c_0 et s_0 en même temps. De la même façon, on peut marginaliser sur une des variables, et dire que l'on a $\frac{5}{7}$ de chance que la sous-classe soit c_0. On peut donc dire de façon informelle qu'une particule est une hypothèse sur l'état courant cru, dans le sens où elle est l'une des possibilités d'affectations complètes des variables.

La phase de propagation consiste alors à déduire par un tirage aléatoire quelles vont être les nouvelles valeurs de chaque variable pour chaque particule. Sur la figure, la particule qui était en haut à gauche, de valeurs (c_0, s_0), devient alors (c_0, s_1), en fonction des probabilités de transition du modèle bayésien. On remarque que dans le cadre des DBNs, les valeurs d'une variable peuvent être influencées par des valeurs du pas de temps précédent et du pas de temps

courant (liens $t \rightarrow t$ et $t \rightarrow (t+1)$). Dans ce cas, une particule est donc une affectation complète des variables sur deux pas de temps consécutifs, et non sur un seul pas de temps comme illustré.

Chaque particule est alors pondérée indépendamment des autres en fonction de la probabilité d'apparition des observations sachant la particule (l'affection complète) et elle uniquement.

Cette pondération sert uniquement à faire un échantillonnage après coup sur les différentes particules, de façon à garder, voire augmenter le nombre des particules qui expliquent le mieux l'observation et diminuer le nombre des "mauvaises" particules qui expliquent mal l'observation. Elle n'entre pas en compte dans le calcul de probabilité d'apparition d'une affectation des variables. Encore une fois, cette information n'est calculée que sur la base du *nombre* de particules prenant telle ou telle valeur pour telle ou telle variable.

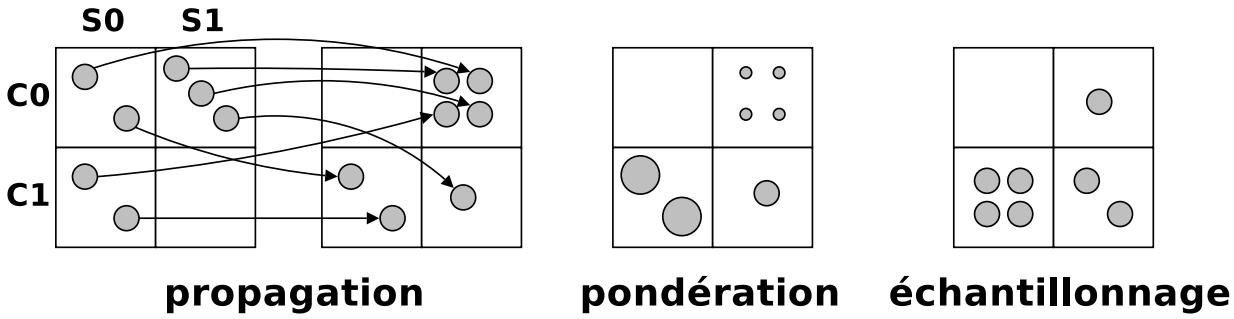


FIG. III.8: Principe du filtrage particulaire

Notons :
- $r_t^{(i)}$ la $i^{\text{ème}}$ particule (qui est donc un vecteur où chaque composante correspond à une valeur d'une des variables cachées) à l'instant t ;
- y_t l'observation faite à l'instant t (et y_t peut aussi être un vecteur de différentes variables observables) ;
- $w_t^{(i)}$ la pondération d'une particule à l'instant t ;
- q est la fonction de transition du DBN, à arguments et valeurs dans l'espace des particules ;
- $A(w_1, ..., w_N)$ est une fonction aléatoire qui renvoie des valeurs de 1 à N proportionnellement aux réels w_1 à w_N.

Au départ, les M particules sont générées en fonction de la distribution a priori sur les variables cachées. A chaque pas de temps, les étapes suivantes sont faites :

1. *propagation* : les particules sont propagées de t à $t+1$ conformément aux quantifications des liens causaux ;
$$\forall i, \qquad r_t^{(i)} = q(r_{t-1}^{(i)})$$

2. *pondération* : chaque particule est pondérée par la probabilité qu'elle donne à l'apparition de l'observation en $t+1$;
$$\forall i, \qquad w_t^{(i)} = P(y_t | r_t^{(i)})$$

3. *échantillonnage* : de nouvelles particules sont créées en tirant aléatoirement parmi les anciennes particules, avec une probabilité de tirage proportionnelle à leur pondération ;

$$\forall i, \qquad \bar{r}_t^{(i)} \leftarrow r_t^{(A(w_t^{(1)},\ldots,w_t^{(M)}))}$$

Cet algorithme permet d'avoir une représentation très compacte d'un état courant cru, mais également très précise et très proche des observations réelles du fait de l'échantillonnage, qui en est encore une fois la partie la plus importante.

Une remarque s'impose : si le nombre de particules est plus grand que le nombre d'hypothèses différentes possibles, on a bien sûr une très bonne approximation de l'état courant cru ; mais ceci reste vrai lorsque le nombre de particules est plus petit voire beaucoup plus petit que la taille de l'espace des hypothèses possibles. En effet, grâce au ré-échantillonage, les particules se concentrent vers les hypothèses les plus probables. Il peut néanmoins arriver à cause des tirages aléatoires que les particules ne représentent plus un état très probable du système, il convient alors d'avoir recours à des aménagements sur l'algorithme principal.

La variante de l'algorithme de filtrage particulaire que nous proposons d'utiliser consiste à faire varier le nombre de particules. On tire d'abord des particules et on regarde la somme des probabilités qu'elles donnent à la première observation (pour avoir un ordre de grandeur de cette probabilité) et, après chaque phase d'échantillonnage, on rajoute une quatrième phase consistant à enlever ou rajouter des particules jusqu'à ce que la probabilité d'avoir l'observation redevienne comparable à celle de départ. Les particules supplémentaires sont tirées à partir des précédentes, mais il peut aussi parfois être intéressant de retirer des particules complètement au hasard, si le DBN contient une sorte d'état absorbant qui fasse tomber trop de particules vers de mauvaises hypothèses (ceci est assez rare et complètement dépendant du processus).

Ces méthodes on été largement étudiées, la variante la plus efficace pour les DBNs est connue sous le nom de "rao-blackwell" [Doucet, Freitas, Murphy et Russell, 2002] qui propose de mélanger inférence approximée et exacte ainsi que d'ajouter du bruit dans l'inférence via une méthode de Markov-Chain Monte Carlo (MCMC), de façon à sortir des estimations trop fortes [MacEachern, Clide et Liu, 1999].

Cette méthode s'est avérée extrêmement efficace pour approximer des distributions de probabilités fortement corrélées de façon compacte, et en gardant une très bonne approximation. Cette méthode s'étend également au cas continu, alors une particule n'est plus discrète, mais peut être tout type de loi de probabilité. Un cas intéressant est celui où une particule est une mixture de gaussienne, ou encore un vecteur de mixtures de gaussienne, car de telles sommes de gaussiennes permettent de représenter tout type de fonction continue sous forme compacte (pour peut que le nombre de gaussiennes soit assez grand). Nous nous restreignons néanmoins au cas discret.

III.3 Apprentissage Quantitatif

L'apprentissage quantitatif d'un DBN $B = (\mathcal{G}, \Theta)$, où $\mathcal{G}$ est la structure du DBN et Θ la paramétrisation associée suivant le critère Q consiste à calculer Θ' tel que $Q(\mathcal{G}, \Theta')$ soit le meilleur

possible. Idéalement, on cherche :

$$\Theta' = \underset{\Theta}{argmax}\, Q(\mathcal{G}, \Theta)$$

Le critère Q le plus courant est la probabilité d'apparition des observations $O = O_{1:T}$ d'entraînement connaissant le DBN :

$$Q = P(O_{1:T}|B) \qquad avec \qquad B = (\mathcal{G}, \Theta)$$

En effet, il est logique de dire qu'un bon modèle est un modèle qui explique le mieux possible les observations rencontrées, c'est-à-dire qui puisse les générer le plus souvent. L'espace des paramétrisations possibles étant dimensions infinies et très nombreuses, on a recours à une recherche locale autour de la paramétrisation Θ du DBN de départ. Du fait de la complexité du critère par rapport aux paramétrisations possibles, il n'est malheureusement pas garanti de trouver un optimum global pour B. Ce problème est exactement le même que pour l'apprentissage des HMMs.

Lorsque l'on a une approximation des états courants crus, c'est-à-dire des corrélations entre toutes les variables à tous les instants, on peut aménager l'algorithme d'apprentissage des HMMs (Expectation-Maximization) pour permettre un apprentissage quantitatif d'un DBN. Nous proposons dans cette section une formalisation originale qui met en évidence l'utilisation de l'état courant cru comme pondération pour trouver les probabilités jointes alors qu'habituellement les probabilités jointes sont calculées directement à partir de l'état courant cru. Le principe général est le même que l'algorithme EM décrit pour les HMMs.

Définitions

On définit la probabilité $\alpha_t(i_j)$ de faire le début d'une séquence d'observations jusqu'à l'instant t et d'avoir une variable q_i prenant la valeur S_{i_j} à l'instant t :

$$\alpha_t(i_j) = P(O_{1:t}, q_t^i = S_j^i|\lambda) \tag{III.3}$$

De la même façon, on définit la probabilité $\beta_t(i_j)$ de faire la fin d'une séquence d'observation en partant d'une valeur S_j^i de la variable q^i en t :

$$\beta_t(i_j) = P(O_{t+1:T}|q_t^i = S_j^i, \lambda) \tag{III.4}$$

On note $\gamma_t(i_j)$ la probabilité que la variable q^i prenne la valeur S_j^i :

$$\gamma_t(i_j) = P(q_t^i = S_j^i|O, \lambda) \tag{III.5}$$

Et finalement, on note $b_{i_j}(O_t)$ la probabilité de faire toutes les observations correctes en t sachant $q_t^i = S_j^i$.

Le gros changement par rapport à un cadre HMM est la définition d'une transition. En effet, dans le cadre HMM il n'y a qu'une seule transition, alors que nous avons maintenant beaucoup

plus de lien causaux, y compris des liens dans le même pas de temps. Une transition est donc définie par une seule variable d'arrivée, et par tous ses parents causaux.

Grâce à cette définition, on définit alors la probabilité de faire l'observation O_t sachant $q_t^i = S_j^i$ et connaissant l'affectation σ_i des parents de q^i :

$$b_{i_j,\sigma_i}(O_t) = P(O_t|q^{Pa(i)} = \sigma_i, q_t^i = S_j^i) \tag{III.6}$$

Algorithmes

Message avant

Maintenant, l'inférence du message avant dépend de $Pa(i)$, et pas uniquement d'une variable q_i à l'instant $t-1$. On calcule alors la probabilité d'arriver dans une instance σ_i des parents de q^i connaissant le début de l'observation $\alpha_t(\sigma_i)$ avec $\sigma_i \in Pa(i)$:

$$\alpha_t(\sigma_i) = P(0_{1:t}, \sigma_i) = [\prod_{i_j \in \sigma_i} \alpha_t(i_j)] B_t(\sigma_i) \tag{III.7}$$

Le coefficient $B_t(\sigma_i)$ (qui dépend aussi de l'historique des observations) doit être introduit car toutes les combinaisons σ_i ne sont pas équiprobables. En effet, comme les variables sont corrélées (par définition du DBN), on doit estimer la probabilité de chaque combinaison σ_i à l'instant t. Cette valeur de $B_t(\sigma_i)$ doit être approximée car un calcul exact nécessite de garder en mémoire tout l'historique. C'est un facteur qui vient directement de l'état courant cru, il nous donne les probabilités des instanciations des parents, en tenant compte des corrélations. C'est précisément cette valeur qui sera approximée par filtrage particulaire tel que décrit dans la section III.2.

Pour le calcul de $b_{i_j}(O_t)$, on se sert de tout σ_i, car l'observation peut être aussi influencée par les parents de i_j. Comme présenté dans [Rabiner, 1989], on obtient :

$$\alpha_t(i_j) = \sum_{\sigma_i \in Pa(i)} \alpha_t(\sigma_i) a(\sigma_i, i_j) b_{i_j,\sigma_i}(O_t) \tag{III.8}$$

Il est à noter ici que l'extension de l'algorithme EM pour les DBN utilisée implicitement habituellement ne fait pas les calculs de cette manière, mais déduit directement les $\alpha_t(i_j)$ du filtre particulaire. Encore une fois, nous parlons d'algorithme implicite pour désigner les travaux sur l'apprentissage quantitatifs des DBNs car a notre connaissance personne n'a explicité la variante pour les DBNs. Nous pensons que notre façon de calculer doit amener à de meilleurs résultats grâce à une plus grande stabilité des calculs, en les rendant moins dépendant du filtre particulaire (en particulier si le nombre de particules choisi est trop faible).

Message arrière

Contrairement au message avant, on obtient une valeur de β en regardant tous les successeurs q^j de la variable dans le DBN. Mais on n'a aucune information sur les autres variables parent de chaque successeur q^j que l'on doit regarder (à cause du sens des liens causaux). On ne peut donc

pas faire d'hypothèse sur les parents de q^j ; on a donc choisi de faire une simple moyenne des calculs des messages arrières sur les différents enfants.

Soit $Ch(i)$ les enfants de la variable i. Donc pour chaque enfant k de la variable i, on construit l'ensemble des σ_k tels que $\sigma_k \in Pa(k)$ tel que $i_j \in \sigma_k$.

On calcule la probabilité de faire la bonne observation à l'instant t multipliée par la probabilité de faire les bonnes observations après t :

$$D_t(k_l) = b_{k_l,\sigma_k}(O_t)\beta_t(k_l) \tag{III.9}$$

On pondère alors cette probabilité par la probabilité d'obtenir k_l lorsque l'on a i_j :

$$E_t(k_l, i_j) = a(\sigma_k, k_l)b_{k_l,\sigma_k}(O_t)\beta_t(k_l) \qquad \sigma_k \in Pa(k) \qquad i_j \in \sigma_k \tag{III.10}$$

On fait ensuite la moyenne sur toutes les affectations des parents possibles et les valeurs l possibles de k ($l \in Va(k)$). Cette moyenne s'écrit :

$$F_t(k, i_j) = \sum_{\substack{\sigma_k \in Pa(k) \\ i_j \in \sigma_k}} \sum_{l \in Va(k)} \frac{a(\sigma_k, k_l)b_{k_l,\sigma_k}(O_t)\beta_t(k_l)}{card(Va(k)).C_k} \tag{III.11}$$

Avec $C_k = card(\{\sigma_k \in Pa(k), i_j \in \sigma_k\})$

D'où $\beta_t(i_j)$ est la somme sur tous les enfants de i_j de cette moyenne, soit :

$$\beta_t(i_j) = \sum_{k \in Ch(i)} F_t(k, i_j) \tag{III.12}$$

Probabilité de faire une transition en un instant donné

On définit la probabilité d'apparition d'une transition en un instant donné (par rapport à la probabilité de transition statique du modèle $a(\sigma_i, i_j)$:

$$\xi_t(\sigma_i, i_j) = P(q_t^{Pa(i)} = \sigma_i, q_t^i = S_{i_j})$$

Par analogie avec l'expression de ξ_t utilisée pour les HMM, on obtient que la probabilité de faire une telle transition connaissant toute l'observation est la probabilité d'arriver en une instance des parents σ_t à l'instant t connaissant le début de l'observation (i.e. $\alpha_t(\sigma_i)$) multipliée par la probabilité de faire la fin de l'observation connaissant la valeur de la variable fille $\beta_t(i_j)$, multipliée par la probabilité de transition du modèle $a(\sigma_i, i_j)$ et la probabilité de faire l'observation à cet instant t $b_{i_j,\sigma_i}(O_t)$:

$$\xi_t(\sigma_i, i_j) = \frac{\alpha_t(\sigma_i)a(\sigma_i, i_j)b_{i_j,\sigma_i}(O_t)\beta_t(i_j)}{P(O|\lambda)} \tag{III.13}$$

Mise à jour du modèle

Il vient alors, exactement comme pour les HMM :

$$a(\sigma_i, i_j) \leftarrow \frac{\sum_t \xi_t(\sigma_i, i_j)}{K} \tag{III.14}$$

avec K facteur de normalisation.

Pour les variables observables n'ayant pas de lien sortant, on ne peut pas calculer de $\beta_t(i_j)$, il faut étendre la définition des HMMs pour les fonctions d'observation. On a alors :

$$\bar{a}(\sigma_i, i_j) = \sum_{t, q_t = S_{i_j}} \frac{\gamma_t(\sigma_{Pa(i)})}{K} \tag{III.15}$$

Algorithme général

L'algorithme général est alors celui de la figure III.9.

1. **tant que** $P(O|\mathcal{G}, \Theta)$ augmente **faire**
2. {Phase d'expectation, c'est-à-dire de complétion de l'observation}
3. **pour** t = 0 à t = T **faire**
4. **pour tous** σ_i **faire**
5. calculer $B_t(\sigma_i)$ connaissant $B_{t-1}(\sigma_j), \forall j$ et O_t, par filtrage particulaire
6. **fin pour**
7. **pour tous** i,j, dans l'ordre de causalité **faire**
8. calculer $\alpha_t(\sigma_i)$ connaissant $\alpha_t(Pa(\sigma_i))$, $\alpha_{t-1}(Pa(i))$, $\alpha_t(Pa(i))$ suivant III.7
9. calculer $\alpha_t(i_j)$ connaissant $\alpha_t(\sigma_i)$ suivant III.8
10. **fin pour**
11. **fin pour**
12. **pour** t = T à t = 0 **faire**
13. **pour tous** i,j, dans l'ordre inverse de la causalité **faire**
14. calculer $\beta_t(i_j)$ suivant III.9,III.10,III.11,III.12
15. **fin pour**
16. **fin pour**
17. {Phase de maximisation de $P(O|\mathcal{G}, \Theta)$}
18. **pour tous** t,i,j **faire**
19. calculer $\xi_t(\sigma_i, i_j)$ selon III.13
20. $a(\sigma_i, i_j) \leftarrow \sum_t \xi_t(\sigma_i, i_j)$ (III.14)
21. **fin pour**
22. $\Theta \leftarrow \{a(\sigma_i, i_j)\}$
23. calculer $P(O|\mathcal{G}, \Theta)$
24. **fin tant que**

FIG. III.9: Algorithme général EM pour des DBNs

Remarque sur la ligne 8 : les $\alpha_t(Pa(i))$ sont nécessaires pour calculer les $\alpha_t(i_j)$. C'est pour cette raison que ce calcul doit être fait dans l'ordre de causalité des variables de façon à ce que dans un même pas de temps, les parents d'une variable aient été calculés avant celle-ci. Le même type de remarque est valable pour le calcul des $\beta_t(i_j)$ à la ligne 14 d'où l'ordre inverse de la ligne 13.

Remarques sur la complexité

Au niveau de la complexité du calcul des $\alpha_t(\sigma_i)$, de nombreux facteurs sont à prendre en compte. On note :

- n_i l'arité de la $i^{\text{ème}}$ variable ;
- N le plus grand des n_i ;
- m_i le nombre de parents de la $i^{\text{ème}}$ variable ;
- M le plus grand des m_i ;
- Q le nombre de variables observables ;
- T le nombre total d'observations.

D'autre part, $n_{Pa(i)}$ est le cardinal de σ_i, c'est-à-dire le nombre d'instanciations des parents d'une variable.

Si l'évaluation de l'état courant cru est en $O(B)$ alors la complexité du calcul de $\alpha_t(\sigma_i)$ est en $O(n_{Pa(i)} + B)$. (Ceci est valable si l'extraction de $\alpha_t(i_j)$ est en $O(1)$, ce qui est extrêmement optimiste, elle est en général en $O(n_i)$. On remarque aussi que B est fonction (linéaire) du nombre de particules qui servent d'estimation pour l'état courant cru, ainsi que de $n_{Pa(i)}$.)

Pour la complexité du calcul des $\alpha_t(i_j)$, il faut estimer la complexité de l'extraction de $a(\sigma_i, i_j)$. Dans l'idéal il est en $O(1)$, mais en général il va être en $O(n_i)$ au mieux. Le calcul de $b_{i_j,\sigma_i}(O_t)$ va dépendre du nombre de variables observables Q et va être au mieux en $O(Q)$ et plus généralement en $O(Q \times n_{Pa(i)})$.

Alors la complexité du calcul d'un seul $\alpha_t(i_j)$ est au mieux en $O(n_{Pa(i)} \times (n_{Pa(i)} + B + Q))$ soit en $O(n^2_{Pa(i)})$. Pour l'ensemble du message avant, on monte alors en $O(n^2_{Pa(i)} \times n_i \times T)$.

Or $n_{Pa(i)}$ est lui-même le nombre d'instanciations possibles des parents de la $i^{\text{ième}}$ variable, alors $n_{Pa(i)}$ est de l'ordre de N^{m_i}. La complexité de cette phase est alors en $O(N^{2m_i} \times n_i \times T)$ pour la $i^{\text{ème}}$ variable. Finalement, pour toutes les R variables, on obtient une complexité du message avant en $O(R \times N^{2M+1} \times T)$.

La complexité du calcul des $\beta_t(i_j)$ est du même ordre de grandeur que pour le message avant (la seule différence est sur le fait qu'on n'estime pas l'état courant cru, mais nous n'en avons pas tenu compte dans la complexité du message avant).

Il vient alors que la complexité d'une phase de mise à jour du modèle est en

$$O(R \times N^{2M+1} \times T^2) \tag{III.16}$$

avec :

- R le nombre total de variables ;
- N l'arité maximale des variables ;
- M le nombre de liens causaux entrants maximal ;
- T le nombre total d'observations.

Implémentation

Une des difficultés principales avec ce type d'algorithme est la même que pour la version pour les modèles de Markov cachés : on manipule des probabilités qui tendent exponentiellement vite vers zéro avec le temps. Il suffit de rappeler la définition du message avant pour le voir :

$$\alpha_t(i_j) = P(O_{1:t}, q_t^i = S_j^i|\lambda)$$

Même si chaque probabilité est grande, la probabilité d'apparition d'une séquence d'observation est de l'ordre de grandeur de p^T, avec $p < 1$. Donc $P(O|\lambda)$ est généralement non représentable sur machine, et ce quelle que soit la précision utilisée, il faudra alors s'intéresser à $log(P(O|\lambda))$ qui lui sera quantifiable. Ici encore un mécanisme de mise à l'échelle est nécessaire.

On adapte la mise-à-l'échelle expliquée dans [Rabiner, 1989]. Dans le cadre HMM, on a un seul facteur de mise-à-l'échelle, attaché à la seule variable cachée. Dans le cadre DBN, on est obligé d'avoir un facteur par variable et par pas de temps, ce qui complique nettement l'implémentation. Ce facteur doit également être appliqué au bon moment. En effet, comme les liens causaux sont beaucoup plus nombreux que pour les HMMs, une même variable va être utilisée plusieurs fois pour calculer des probabilités pour différentes variables. Calculer des probabilités à la fois en utilisant des variables mises à l'échelle et d'autres qui ne le sont pas conduit à des calculs faux. Il faut faire bien attention à calculer d'abord tous les α pour toutes les variables pour un pas de temps, puis tout mettre à l'échelle, et ne pas mélanger ces deux opérations. Le problème est alors qu'en général on se sert de variables de plusieurs pas de temps différents pour calculer une probabilité, et alors le pas de temps précédent a été mis à l'échelle et pas le pas de temps courant. Il faut prendre beaucoup de précautions lors de l'implémentation, car si le facteur de mise à l'échelle est mal appliqué, les calculs n'ont aucune chance de converger.

Dans l'algorithme précédent, on calcule donc :

$scale(i,t) = \sum_j \alpha_t(i_j)$
pour tous j **faire**
    $\alpha_t(i_j) \leftarrow \frac{\alpha_t(i_j)}{scale(i,t)}$
fin pour

que l'on insère entre la ligne 9 et 10.

De la même façon que pour les HMMs, on utilise le même facteur d'échelle pour les $\beta_t(i_j)$, et on insère entre la ligne 14 et 15 :

pour tous j **faire**
    $\beta_t(i_j) \leftarrow \frac{\beta_t(i_j)}{scale(i,t)}$
fin pour

III.4 Apprentissage Structurel

Maintenant que nous avons vu comment adapter l'apprentissage quantitatif (des probabilités) des HMMs vers les DBNs, penchons-nous sur l'apprentissage de la structure pour les réseaux bayésiens dynamiques. Si l'on a un DBN $B = (\mathcal{G}, \Theta)$, où $\mathcal{G}$ est la structure du DBN et Θ la

paramétrisation associée, suivant le critère Q, l'apprentissage structurel consiste à calculer $\mathcal{G}$ tel que $Q(\mathcal{G}, \Theta')$ soit le meilleur possible. Idéalement, on cherche :

$$\mathcal{G}' = \underset{\mathcal{G}}{argmax}\, Q(\mathcal{G}, \Theta)$$

Dans $\mathcal{G}$ sont spécifiées à la fois la structure des connections causales entre variables, mais aussi l'arité des variables. Cette dernière composante est extrêmement dépendante du domaine, et peu de travaux ont vu le jour sur cet aspect. Ceci est dû au fait que cette question ne se pose pas dans le cas où toutes les variables sont observables, et que pour les variables cachées, apprendre uniquement la structure causale est déjà très difficile. Le genre d'approche que nous avons développée dans le cadre des HMMs pourrait très bien s'appliquer à ce problème d'arité.

Nous allons présenter différentes techniques de l'état de l'art en les replaçant dans leurs contextes spécifiques, et citerons simplement une méthode très simple que nous avons utilisée.

Apprentissage de Réseaux Bayésiens

Un réseau bayésien dynamique est un réseau bayésien, et plusieurs méthodes ont été proposées pour apprendre la structure des réseaux bayésiens, hors des considérations propres aux DBNs. Il faut noter que l'apprentissage d'un réseau bayésien optimal par rapport à des données (i.e. retrouver la structure exacte qui a servi à générer les données) est un problème NP-complet [Chickering, 1996].

Plusieurs famille de techniques ont été proposées pour répondre à ce problème. La première (IC [Pearl, 2000], PC [Spirtes, Glymour et Scheines, 1993]), consiste à déterminer dans un premier temps un graphe non orienté tel que toute variable est indépendante des variables qui ne lui sont pas adjacentes conditionnellement à un sous-ensemble X du reste des variables, puis à orienter ce graphe pour obtenir un réseau bayésien. La détermination des ensembles X étant exponentielle, ces algorithmes sont peu efficaces pour des problèmes de grande taille.

La deuxième famille (K2, [Cooper et Herskovits, 1992], SEM [Friedman, 1998]) procède par une recherche locale dans l'espace des réseaux bayésiens, après y avoir défini un voisinage et une fonction de score (généralement liée à la complexité du réseau). Le voisinage entre réseaux est simplement défini par l'ajout/retrait d'un lien causal, et le score est en général simplement le nombre lien causaux, ou plus finement la quantité d'information nécessaire pour coder l'ensemble du réseau (dans ce cas un lien causal vers une variable de grande arité sera plus coûteux qu'un lien causal vers une variable de petite arité). Toutefois, la présence dans le voisinage de beaucoup d'autres réseaux équivalents (qui ont les mêmes indépendances) conduit à de nombreuses évaluations de la même classe d'équivalence et génère de multiples optima locaux.

La troisième famille (GES [Chickering, 2002], EQ [Munteanu et M.BEndou, 2001] pour pallier cela, déplace la recherche dans l'espace des graphes essentiels, qui est l'espace des classes d'équivalences des réseaux bayésiens. Cet espace peut être parcouru sans rencontrer d'optima non globaux mais nécessite des opérateurs de voisinages nombreux et complexes pour passer d'une classe d'équivalence à une autre.

Un algorithme a été proposé très récemment pour faire une recherche dans un autre espace proche des graphes essentiels, et qui permet de diminuer grandement la complexité de la recherche [Gonzales et Jouve, 2006].

Données Incomplètes

Plusieurs problèmes se posent pour appliquer directement ces algorithmes aux réseaux bayésiens dynamiques. Le principal d'entre eux est que nous traitons le cas de variables cachées, ce qui n'est pas le cas de ces algorithmes. En effet, ils se basent sur des statistiques jointes sur les variables, qui ne sont évidemment pas disponibles dans le cas de variables cachées. Il faut alors dans un premier temps compléter les observations pour rendre toutes les variables observables (en calculant les γ définis précédemment, c'est-à-dire les probabilités pour les variables cachées tout au long du temps) et ensuite faire un apprentissage de structure, ce qui est très lourd au niveau calculatoire.

Dans notre application, la structure générale est relativement connue, comme nous allons le voir dans la section III.6. Nous sommes simplement partis d'un DBN avec presque tous les liens causaux possibles, puis avons essayé de supprimer des liens causaux en utilisant l'information mutuelle entre variables liées par des liens causaux de façon à supprimer certains liens.

La recherche de structure avec variables cachées est néanmoins un domaine complexe où de nombreux travaux ont été menés. Les recherches locales sont adaptées pour contourner la complexité : l'algorithme SEM [Friedman, 1998] propose de mélanger en profondeur recherche locale de structure et apprentissage quantitatif, mais comme nous l'avons déjà dit, il est lourd et surtout ne permet d'atteindre que des optima locaux. On se retrouve alors avec une recherche locale de structure greffée sur une recherche locale de probabilités, ce qui ne saurait conduire qu'à des optima locaux. Mais même ainsi, un deuxième problème se révèle très dur à contourner : nous avons besoin de données statistiques sur les variables, et donc il faut un très grand nombre d'observations pour effectuer la recherche. Or dès que le DBN est grand, il est très dur de couvrir l'espace des observations dans une expérimentation réelle.

De plus, le dernier problème est intrinsèque à la non-observabilité : en effet, il est très dur de déterminer avec précision le nombre et l'arité des variables cachées qui va permettre d'apprendre de bons modèles du système, alors que ces paramètres sont indispensables pour obtenir un DBN cohérent avec le processus à modéliser. Dans [Elidan, Lotner, Friedman et Koller, 2000], les auteurs proposent une méthode pour ajouter des variables cachées en se basant sur la structure actuelle du DBN. Ils proposent d'ajouter des variables lorsque la structure courante est trop complexe, en vue de la simplifier. Ils proposent une recherche basée sur des semi-cliques reliant trop de variables observables. Pour trouver l'arité des variables cachées, les auteurs de [Elidan et Friedman, 2001] proposent une méthode inspirée des méthodes de classification agglomératives et de l'assemblage de modèles bayésiens. Ce point reste néanmoins très compliqué à résoudre dans le cas général. Le problème de la recherche d'une structure optimale étant donné un ordre des variables a été traité efficacement dans [Friedman et Koller, 2003], et ce cas peut être très intéressant pour un grand nombre d'application réelles. Les techniques d'évaluation des réseaux bayésiens sont particulièrement bien détaillées dans [Friedman, Murphy et Russell, 1998], ainsi que les problèmes qui se posent pour le cas à variables cachées. Une technique basée sur la notion d'information mutuelle pour trouver les liens entre variables est présentée dans [Friedman, Nachman et Peér, 1999]. Dans [Elidan, Ninio, Friedman et Schuurmans, 2002], on peut voir une technique de perturbation des données d'apprentissage en vue de sortir des optimas locaux, aussi bien pour l'apprentissage structurel que quantitatif.

Finalement, plusieurs de ces approches sont unifiées élégamment dans [Elidan et Friedman, 2005]. Les auteurs proposent une recherche suivant deux critères très différents. L'idée est qu'une variable cachée doit servir à capturer l'information significative des données d'entraînement sous une forme plus simple. Deux réseaux bayésiens sont alors appris, l'un qui cherche à classifier l'information pour ne retenir que les informations significatives, l'autre capture les probabilités de générer l'observation connaissant uniquement la variable cachée. Les deux critères antagonistes sont alors la compression maximale des données et la meilleure explication de ces données. En partant d'une compression maximale et en la diminuant au fur et à mesure, le nombre de valeurs utilisées de cette variable augmente. L'effet le plus important est que cette méthode permet de s'échapper des optimas locaux d'un apprentissage standard. Cette approche se généralise à plusieurs variables cachées, et à la recherche locale de structure en incluant des ajouts/retraits de liens causaux dans la boucle. Pour déterminer l'arité optimale d'une variable, il suffit d'ajouter une valeur possible lorsque toutes sont utilisées. L'ajout de nouvelles variables cachées peut aussi être fait lorsque la compression diminue. Toutes ces phases sont clairement intégrées dans le même cadre mathématique dans cet article. Cette méthode semble extrêmement prometteuse, mais est très coûteuse en temps d'apprentissage, même pour des problèmes de taille moyenne.

III.5 Apprentissage en ligne ?

Le Problème

Comme nous l'avons souligné, l'apprentissage d'un bon DBN (qui explique bien les données) peut être extrêmement lourd. Pour utiliser ce type de modèle sur un robot, il peut être problématique que l'apprentissage prenne plusieurs jours pour quelques heures (voir minutes) de données. De plus, classiquement l'apprentissage est fait à partir d'un point de départ peu informé (en général aléatoire), car toutes les observations doivent être prises en compte en même temps comme nous allons l'expliquer. Donc apprendre un DBN est de plus en plus long avec la quantité de données recueillies.

Pistes

Comme nous l'avons déjà souligné, l'apprentissage de réseau bayésien dynamique est une recherche locale, et tant que l'on n'utilise pas les techniques récentes pour sortir des optimas locaux, les phases d'apprentissage vont s'arrêter sur des solutions locales. Or l'espace des DBNs possibles est de grande dimension et de forme particulièrement complexe. Ceci est un problème pour l'apprentissage incrémental. En effet, si l'on apprend un DBN pour une portion des observations O_x, la forme de l'espace des DBNs de bonne qualité pour ces observations E_x est a priori très différente de celle de l'espace des DBNs E_y de bonne qualité pour un autre jeu d'observations O_y. Dans ce cas, même si l'on a deux très bons modèles pour chacun des jeux d'observations O_x et O_y, il est très peu probable que la somme des deux soit un bon DBN pour l'union des observations $O_{x \cup y}$. En effet, rien ne nous garantit que l'espace des bons DBNs $E_{x \cup y}$

pour l'union des observations $O_{x \cup y}$ soit lié de façon simple aux deux espaces E_x et E_y des bons DBNs pour les deux parties de l'observation.

Nous avons expérimenté sur le DBN de notre application robotique, présenté dans la section III.6, en découpant les observations en 10 portions égales, et en apprenant le DBN total en 10 fois sur ces dix ensembles d'observations. Le DBN courant est λ_1, le nouveau DBN appris sur une nouvelle tranche d'observations est λ_2. Nous avons expérimenté plusieurs types de sommations des DBNs :

1. la première sommation de deux DBNs λ_1 et λ_2 est une sommation simple, chaque probabilité condionnelle de λ_2 est sommée à celle correspondante de λ_1, le résultat est ensuite évalué par le calcul de $P(O|\lambda_{1+2})$, où O est l'ensemble des observations ;

2. pour la deuxième, on introduit un facteur de pondération en plus pour chaque probabilité ; ce facteur est le nombre de fois où la probabilité a été mise-à-jour par le processus d'apprentissage ;

3. pour la troisième, on se sert du facteur de pondération introduit précédemment, mais de façon binaire : la somme des probabilités de chaque DBN est pondérée par 1 ou 0 si les probabilités correspondantes on été apprises ou non.

Pour chacune de ces sommations, on utilise deux variantes dans le processus itératif. Dans le premier cas, on ne fait rien de spécial par rapport à ce qui a été décrit (on se sert de la somme des DBNs pour la suite du processus), alors que dans le deuxième cas, on mémorise à chaque fois le DBN et si après une itération la nouvelle somme n'est pas meilleure, on repart du meilleur DBN en utilisant le jeu d'observations suivant. De cette façon, si un "DBN somme" est plus mauvais que lors de l'itération précédente, on l'oublie et on repart du meilleur.

La figure III.10 montre les résultats en utilisant la première somme, la figure III.11 la deuxième, et la figure III.11 la troisième.

Une première conclusion est que repartir du meilleur DBN n'est pas une bonne idée, car cela n'amène pas à un meilleur DBN final. Ceci est rassurant car cela consisterait à apprendre un DBN sur une partie des données et pas sur toutes. Un autre fait marquant avec ces courbes est leur non-monotonie. Ces faits s'expliquent par la complexité des espaces des bons DBNs, et montrent qu'on ne peut a priori pas trouver de méthode simple pour l'apprentissage incrémental. Rien ne garantit avec ce genre de procédure que le DBN somme sera meilleur que les DBNs précédents.

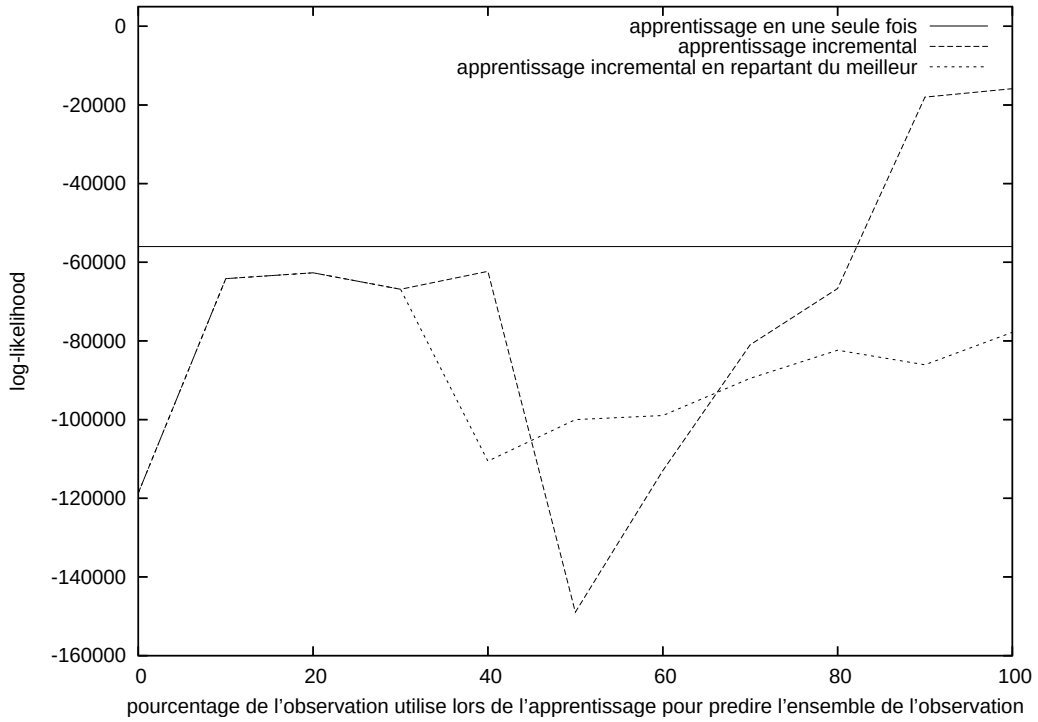


FIG. III.10: Sommation simple des DBNs

III.6 Application

Dans cette application, plusieurs objectifs sont présents :

- améliorer le comportement global du robot par rapport à un critère objectif, par exemple diminuer l'apparition de cas d'échecs ;
- rendre son comportement compréhensible s'il est en présence d'humains ;
- comprendre son fonctionnement interne.

De nombreux travaux se sont penchés sur la possibilité d'avoir des comportements robotiques intuitivement interprétables. En particulier, dans [Oates, Schmill et Cohen, 2000] les auteurs proposent une méthode de classification non supervisée de tâches robotiques (navigation et saisie d'objet) qui donne des classes facilement interprétables par l'humain. Notre approche est très différente dans le sens où nous voulons que lors de la phase d'apprentissage, un opérateur donne des indications qualitatives de comportement sur le comportement du robot, dans le but de pouvoir ensuite choisir entre ces comportements (voir chapitre V). Dans un cadre d'interaction homme-robot, nous nous sommes intéressés à l'adéquation du comportement du robot par rapport à des personnes autour de lui, afin de rendre son comportement de navigation lisible et ergonomique.

Nous avons largement utilisé le cadre des réseaux bayésiens dynamiques pour modéliser le comportement d'un robot lors de sa navigation autonome. Le schéma fonctionnel de la modalité de navigation utilisée est visible sur la figure III.13(a). Elle se base sur une ré-implémention de la technique d'évitement réactif décrite dans [Minguez et al., 2004]. On peut voir une photo du

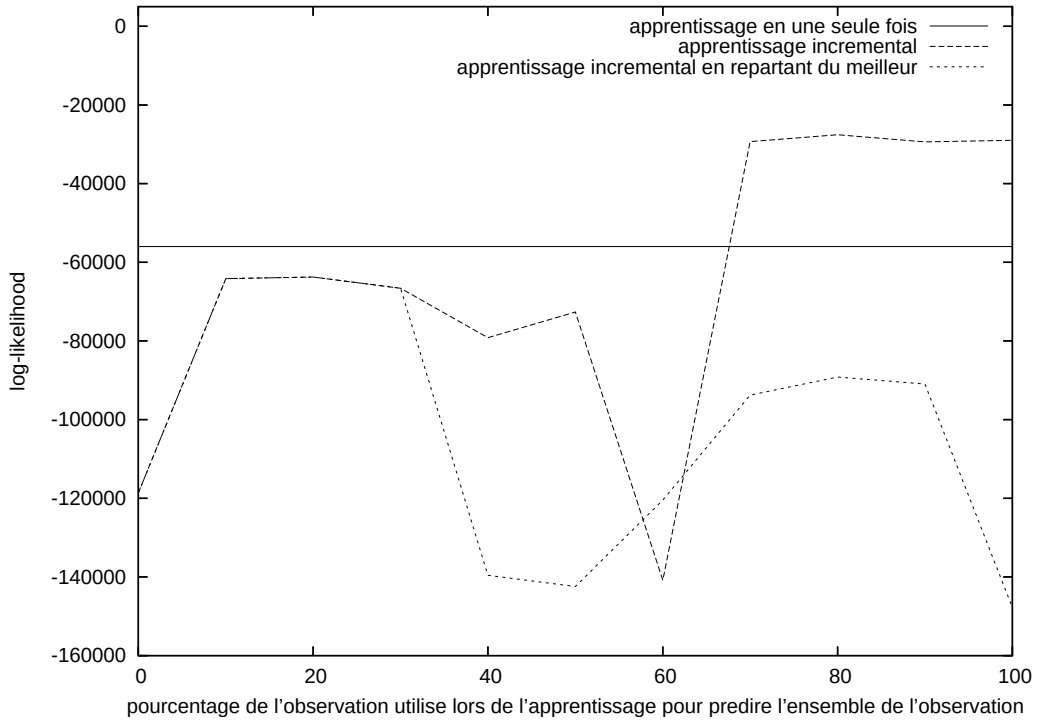


FIG. III.11: Sommation pondérée par le nombre de mises-à-jour lors de l'apprentissage

robot *rackham* qui a servi de plate-forme expérimentale. Ce robot sert de guide de musée [Clodic, Fleury, Alami, Herrb et Chatila, 2005].

Pour cette application, de nombreuses données sont observables, comme l'environnement dans lequel évolue le robot, ainsi que les paramètres de contrôle de l'évitement réactif, qui est au cœur de la navigation. La structure est assez bien connue, du moins d'un point de vue général. Modéliser ce système sous forme de modèle de Markov caché impliquerait alors une perte de connaissance, car il faudrait se priver de nombreuses variables observables, et construire un modèle de Markov caché par jeu de paramètres, comme expliqué sur la figure III.6. On ne se sert de variables cachées que pour modéliser le comportement proprement interne (nous y reviendrons).

Il est à noter que ce type d'approche peut être comparé à certains travaux d'apprentissage d'opérateur pour la planification tels que [Levine et DeJong, 2006]. En effet, de par la nature même d'un DBN, on peut interpréter les différentes transitions possibles de valeurs d'une variable à une autre comme autant de préconditions, et les valeurs post-transition des actions comme des effets d'un action. Notre approche généralise donc l'apprentissage d"opérateurs planification au cas probabiliste, et permet de dégager des variantes d'une même action en accord avec un jugement humain extérieur. Nous reviendrons sur les aspects prise de décision à partir de ce type de modèle dans le chapitre V.

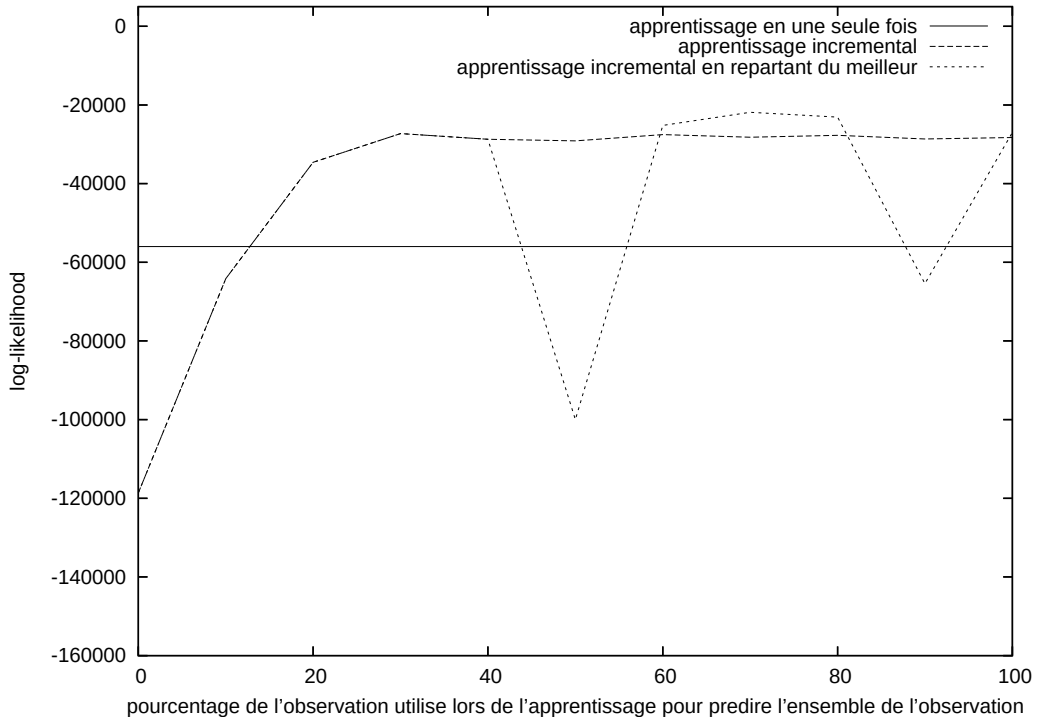


FIG. III.12: Sommation pondérée binairement

Données

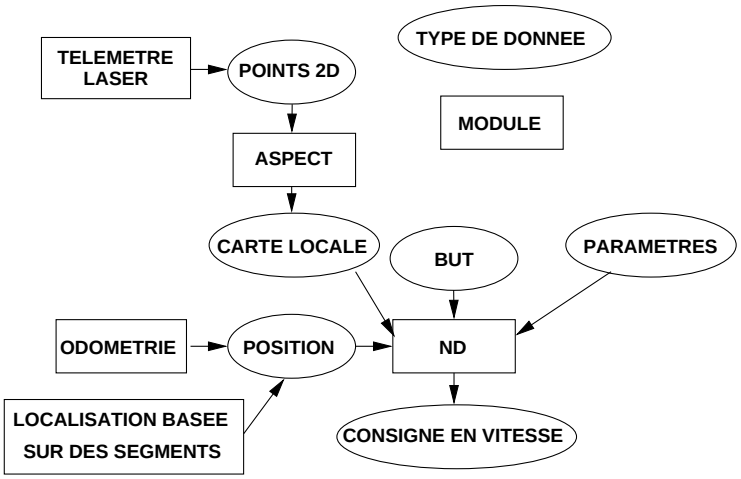
Données brutes

De par l'implémentation des différents modules fonctionnels sur le robot, de nombreuses données brutes sont disponibles. En particulier, les modules fonctionnels exportent de nombreuses données exploitables. Ces modules sont :

- "sick", qui gère le télémètre laser ;
- "pom", qui gère l'estimation de la position courante ;
- "aspect", qui maintient une carte d'aspect, c'est-à-dire de premier plan autour du robot ;
- et "ND", qui calcule les consignes en vitesse que le robot doit exécuter, en fonction des obstacles.

Pour le télémètre laser, on dispose bien entendu de l'ensemble des points obstacle détectés, et il va falloir en extraire une information plus simple à utiliser que les 360 distances brutes qui sont renvoyées à la construction de carte, ceci en fonction de ce que l'on connaît du fonctionnement de ND. Il donne également une représentation en segments et pas uniquement en nuages de points.

Le module aspect qui gère une carte locale autour du robot nous donne les mêmes informations que le télémètre laser, mais sur 360 degrés, alors que le télémètre laser ne nous donne que des informations sur l'avant du robot (tous les demi-degrés). Il peut être considéré comme une mémoire donnant les même informations qu'un télémètre laser à 360 degrés.



(a) Modalité de navigation basée sur l'évitement réactif ND

(b) Plate-forme expérimentale

FIG. III.13: Rackham : Un RWI B21R modifié pour faire guide de musée

Le module PoM (POsition Manager) donne l'estimation des coordonnées courantes du robot, ainsi que les incertitudes, en se servant de l'odométrie et d'une localisation basée sur les segments issus du télémètre laser, mis en correspondance avec une carte statique de l'environnement.

Le module de navigation ND est une navigation réactive qui se base sur la présence d'obstacles dans plusieurs zones de sécurité autour du robot. Il commence par découper l'environnement secteurs angulaires (appelés vallées) qui sont des zones sans discontinuités et suffisamment profondes pour que le robot puisse y avancer. Il choisit ensuite la vallées qui est angulairement la plus proche du but. Ensuite une analyse est faite pour voir si des obstacles sont dans des zones de sécurité autour du robot, indépendamment des vallées. Puis en fonction de cette analyse et de la largeur de la vallée choisie, différentes stratégies sont possibles pour sélectionner un secteur angulaire "but". Finalement, des consignes en vitesse linéaire et angulaires sont générées en fonction du secteur angulaire vers lequel le robot doit se diriger et de la présence d'obstacles proches [Minguez et al., 2004].

Le module ND lui-même nous permet d'avoir accès aux paramètres de contrôles qui sont :

– la largeur maximale du robot : un cercle représente le robot dans les algorithmes de ND ; si un passage entre obstacles est plus étroit que le diamètre de ce cercle, alors ND ne construit

pas de vallée pour ce passage ;
- la distance de sécurité : une demi-ellipse de petit diamètre la largeur du robot et de grand diamètre cette distance de sécurité fixe une zone de sécurité dans laquelle tout obstacle est à éviter ; quand un obstacle est dans cette ellipse, seules les rotations sont autorisées de façon à faire sortir l'obstacle de cette ellipse (vitesse linéaire nulle) ; on peut voir une illustration de ces deux paramètres sur la figure III.14 ;
- la vitesse linéaire maximale autorisée : ND va donner une vitesse linéaire comprise entre 0 et cette valeur ; de plus, cette vitesse linéaire maximale sert à donner une zone dans laquelle les obstacles sont pris en compte ;
- la vitesse angulaire maximale autorisée : ND va donner une vitesse angulaire comprise entre cette valeur et son opposé ;
- le facteur de linéarité : les vitesses linéaires et angulaires données par ND sont proportionnelles à l'angle choisi pour le prochain mouvement, élevé à une certaine puissance ; c'est cette puissance que l'on peut fixer.

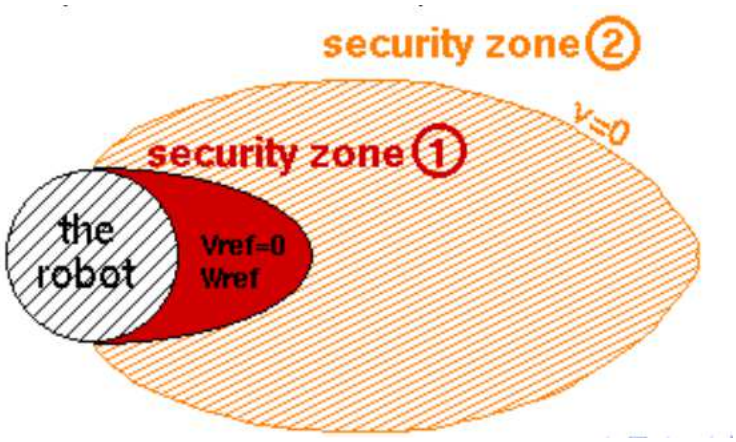


FIG. III.14: Illustration des distances de sécurité utilisées par ND (la zone de sécurité 2 est définie en fonction de la vitesse maximale du robot, la zone 1 en fonction de la distance de sécurité)

ND donne en outre les consignes en vitesse (linéaire et angulaire) que doit exécuter le robot, ainsi que le but en coordonnées locales au robot et le nombre de vallées qu'il construit en interne. Ce nombre de vallées, donc de passages possibles, est une caractéristique importante de l'environnement, ainsi que leur largeur. La stratégie courante est une indication importante de ce qu'est en train de faire l'évitement réactif.

Finalement, on dispose aussi des indications qualitatives fournies par l'opérateur. En effet, on demande lors de la collecte de données à celui-ci de donner une indication sur le comportement du robot du point de vue de son ergonomie pour les personnes qui circulent autour de lui. Le robot peut en effet avoir un comportement adéquat, ou bien éventuellement trop agressif s'il a tendance à avoir des accélérations ou des décélérations trop brusques, qui peuvent rendre sa présence inconfortable, voire désagréable. Il peut à l'inverse avoir un comportement trop timide, trop lent pouvant provoquer une certaine impatience d'un humain qui doit l'attendre. De la même

façon, le type de trajectoires qu'il utilise peut sembler très compréhensible et intuitif, ou au contraire stupide si le robot a tendance à faire des détours ou à prendre des trajectoires très larges. Ces différents types de comportements sont nettement observables avec différents jeux de paramètres du module ND.

Sélection des données

Le choix des données vise à satisfaire deux critères principaux : essayer de capturer le maximum d'information pertinente suivant ce que l'on sait du fonctionnement de la navigation autonome présentée, et le besoin de compacité de façon à pouvoir apprendre un modèle en un temps raisonnable. Ces deux critères sont complètement antagonistes, et trouver le juste équilibre ne peut se faire malheureusement que par essais-erreurs.

Paramètres de contrôle :

- Pour le grossissement du robot, on groupe les deux paramètres de contrôle qui sont la largeur réelle du robot, et la longueur de la zone de sécurité. Pour l'utilisation habituelle du robot, une largeur de 0.6 mètre et une distance de sécurité de 0.7 mètre sont utilisées. Cette largeur correspond à peu près à la largeur réelle du robot, et la distance de sécurité de 0.7 s'est avérée plutôt bien adaptée aux environnements dynamiques moyennement encombrés, mais pas forcément pour des environnements statiques très fortement contraints. Nous avons également utilisé le couple $(0.6; 0.5)$[‡], ainsi que $(0.8; 0.7)$ et $(1.0; 1.0)$[§].
- Pour la vitesse linéaire maximale autorisée, la valeur habituelle est de 0.3 mètre/seconde, nous avons également utilisé 0.2, 0.4 et 0.5, de façon à pouvoir identifier l'influence de la vitesse sur les comportements.
- La vitesse angulaire maximale habituellement utilisée est de 0.2 radians/seconde, nous utilisons également 0.1, et 0.3.
- Le facteur de linéarité nous donne le degré de la courbe v/ω où v est la consigne en vitesse linéaire, et ω la consigne en vitesse angulaire. Dans des versions précédentes de ND, un facteur de 1 était imposé en dur. Nous permettons également 0, 3 et 5. 0 signifie que v et ω ne sont pas liées entre elles. Plus exactement, $v/\omega = k\frac{1-G}{G}$, et $G = \frac{\theta}{\pi/2}^{l}$, avec k une constante dépendant de la stratégie choisie, θ un secteur angulaire "but" choisi suivant la stratégie également, et l le facteur de linéarité paramètre.

Paramètres d'environnement :

- Les données issues du télémètre laser ne sont pas exploitables telles quelles, il faut utiliser des fonctions pour en extraire des indications exploitables sur l'environnement. La première d'entre elles permet de mesurer l'encombrement devant le robot. Cette variable est la même que pour les HMMs du chapitre précédent. Son principe est illustré sur la figure II.4.
- Une autre fonction sur les données brutes du télémètre laser sert à discriminer des cas d'environnements complexes (zone large avec beaucoup d'obstacles) par rapport à des

[‡]que nous espérons plus efficace en environnement fortement encombré quasi-statique

[§]que nous espérons plus efficaces pour des grandes vitesses en environnement peu contraint

cas d'environnements plus simples comme des couloirs, alors que la fonction précédente ne permet pas de discriminer ces deux cas. Cette fonction somme les différences entre les angles des droites porteuses des différents obstacles, comme illustré sur la figure III.15. Ces

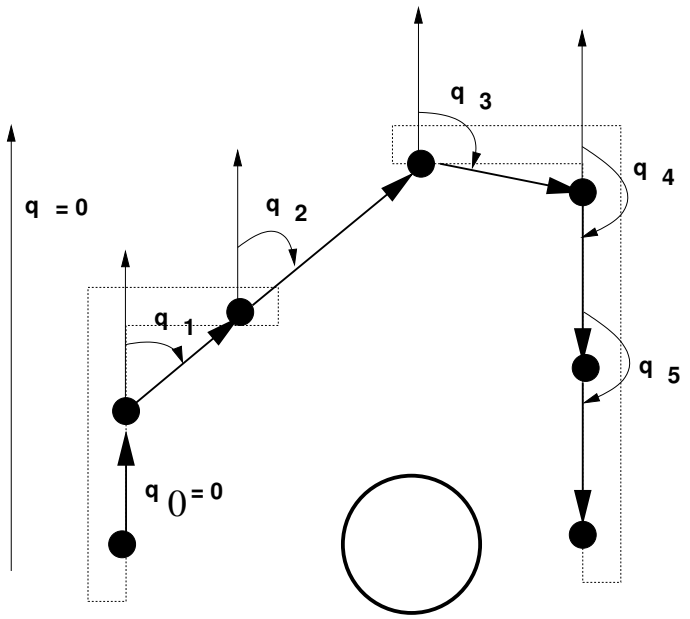


FIG. III.15: Principe de l'évaluation de la complexité géométrique d'un lieu

deux fonctions ont été développées et utilisées avec succès par Morisset dans [Morisset, 2002]. Cependant, dans notre application, les valeurs de cette variables changeaient peu car nous n'avons jamais expérimenté dans un couloir. Nous avons donc abandonnée son usage.

– Une autre donnée capitale pour bien capturer le fonctionnement de ND concerne l'obstacle le plus proche, qui va fortement influencer le déplacement du robot. Une donnée importante est donc la distance du robot à cet obstacle.
– De la même façon, on utilise la position de cet obstacle le plus proche relativement au robot.
– On utilise également le nombre de segments autour du robot, qui est aussi un bon indicateur de la complexité de l'environnement.
– Le nombre de vallées construites par ND nous donne également un bon indicateur de la complexité de la scène.

Paramètres observables conséquence du fonctionnement de ND :

– les consignes données directement par ND : vitesse linéaire et vitesse angulaire référence ;
– les accélérations induites par ces consignes, c'est-à-dire les différences entre deux mesures successives des vitesses linéaire et angulaire ;
– pour l'avancement de la mission : la distance au but estimée par une distance euclidienne, et surtout la différence de cette valeur entre deux pas de temps ;

– une estimation du pourcentage effectué de la mission, (en approximant les distances par des distances euclidiennes) ;
– et finalement, la stratégie choisie par ND, qui devrait nous permettre de prévoir les accélérations.

Paramètres généraux :
– On dispose du comportement général, parmi début, fin (et succès), échec ou rien à signaler. Cette variable est renseignée automatiquement par le processus de collecte des données.
– Et enfin, pour pouvoir dégager des comportements qualitativement et visuellement différents, on demande à l'expérimentateur de donner à la main au cours des expérimentations la valeur d'une variable d'adéquation du comportement du robot par rapport aux personnes aux alentours. Les valeurs peuvent être "correct", "trop timide" ou "trop agressif". "Trop timide" représente le cas où le robot est trop lent et ennuyeux, (intuitivement : trop prudent), alors que "trop agressif" sert à couvrir les cas où les accélérations ou les vitesses sont trop grandes, et des personnes ont peur d'être heurtées par le robot ou que le robot heurte un obstacle.

Les variables continues discrètes sont discrétisées après collecte des données par un algorithme *k-means* (voir II.2), en 5 classes différentes. Notre implémentation requiert des variables à valeurs discrètes. En résumé, on obtient le tableau III.1.

nom	type	val. 1	val. 2	val. 3	val. 4	val. 5	unité
élargissement	C	(0.6 ; 0.7)	(0.6 ; 0.5)	(0.8 ; 0.7)	(1.0 ; 1.0)		mètres
v lin. max	C	0.3	0.2	0.4	0.5		$m.s^{-1}$
v. ang max	C	0.1	0.2	0.3			$rad.s^{-1}$
facteur de linéarité	C	1	0	3	5		
encombrement	E	< 0.7	$[0.7; 1.4]$	$[1.4; 2.0]$	$[2.0; 2.8]$	> 2.8	mètres
distance à l'obstacle	E	< 0.7	$[0.7; 1.1]$	$[1.1; 1.6]$	$[1.6; 2.1]$	> 2.1	mètres
angle de l'obstacle	E	$[-90; -45]$	$[-45; 0]$	$[0; 45]$	$[45; 90]$		degrés
nombre de segments	E	< 11.9	$[11.9; 18.1]$	$[18.1; 24.4]$	$[24.4; 31.0]$	> 31.0	
nombre de vallées	E	≤ 2	$[3; 5]$	$[6; 8]$	$[9; 10]$	> 10	
vitesse linéaire	O	< 0.07	$[0.07; 0.19]$	$[0.19; 0.31]$	$[0.31; 0.42]$	> 0.42	$m.s^{-1}$
vitesse angulaire	O	< -0.15	$[-0.15; -0.05]$	$[-0.05; 0.05]$	$[0.05; 0.15]$	> 0.15	$rad.s^{-1}$
accélération linéaire	O	< -0.2	$[-0.2; -0.05]$	$[-0.05; 0.05]$	$[0.05; 0.15]$	> 0.15	$m.s^{-2}$
accélération angulaire	O	< -0.3	$[-0.3; -0.08]$	$[-0.08; 0.08]$	$[0.08; 0.3]$	> 0.3	$rad.s^{-2}$
Δ dist. au but	O	< -0.3	$[-0.3; -0.16]$	$[-0.16; -0.05]$	$[-0.05; 0.02]$	> 0.02	mètres
pourcentage	O	< 12	$[12; 33]$	$[33; 54]$	$[54; 76]$	> 76	%
stratégie	O	HSWV	HSNV	HSGV	LS1	LS2	
général	Q	début	succès	échec	rien		
adéquation	Q	compréhensible	agressif	timide			

TAB. III.1: Liste des variables observables du DBN (C : contrôlable, E : environnementale, O : observable simple, Q : qualité)

Structure

La structure du DBN est connue dans ses grandes lignes. L'environnement et les paramètres de contrôle influencent causalement le module de génération de mouvement, qui sera alors modélisé par des variables cachées. Cet état interne caché influence quant à lui les variables observables que sont le mouvement du robot, et son comportement général et observable. Donc les

grands groupes d'influence sont connus. Néanmoins la présence de chaque lien causal est individuellement inconnue (est-ce que la vitesse maximale influence la seconde variable cachée ? nous n'en savons rien). On utilise donc une structure complète pour chacun des groupes d'influence. Ceci nous donne la structure générale de la figure III.16.

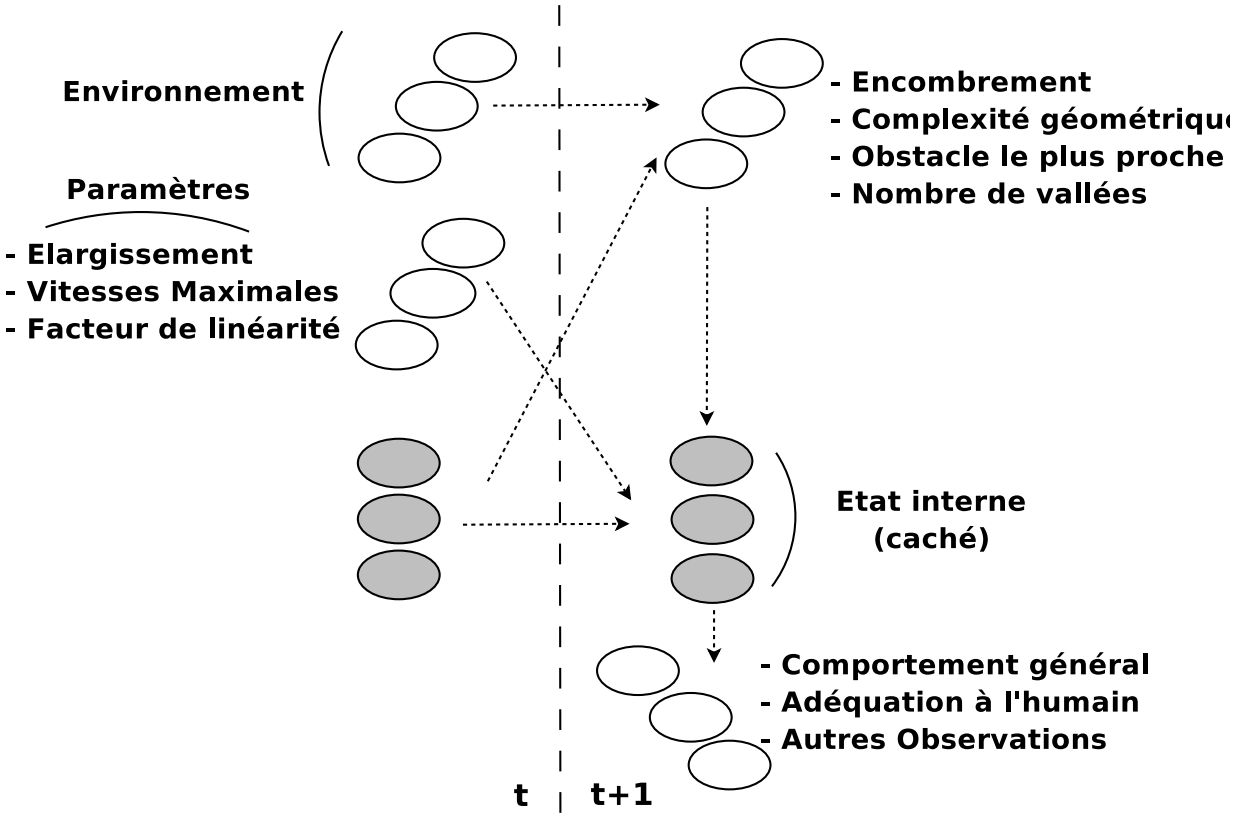


FIG. III.16: Structure générale du DBN pour la modélisation de la navigation autonome

Résultats

Nous avons collecté des données sur 118 navigations autonomes du robot. Les données sont collectées toutes les 600 millisecondes, c'est-à-dire à une période légèrement supérieure à celle de ND qui refait tous ses calculs et produit une nouvelle consigne en vitesse toutes les 400 ms ; ceci de façon à être sûr de ne pas collecter deux fois la même observation au niveau du processus de génération de mouvement. Le nombre d'observations est de plus de 7 500, pour plusieurs centaines de mètres de navigation. L'apprentissage du modèle est ensuite fait hors-ligne.

L'apprentissage du modèle est assez long du fait du grand nombre de liens causaux que nous avons utilisé dans notre structure. Une itération de EM pour les DBNs prend plusieurs minutes, et la convergence vers un optimum local du modèle prend de l'ordre de la dizaine d'itérations. Sur un pentium® 4 3 GHz, l'apprentissage du modèle prend un peu plus d'une heure. Ce calcul est

donc un peu lourd pour l'embarquer directement sur un robot, à moins que le robot ne s'arrête de fonctionner pendant la phase d'apprentissage proprement dite. Ceci n'est pas forcément gênant car la collecte des données et leur stockage n'est pas lourd, et donc on peut se servir des phases de "repos" du robot pour apprendre le modèle.

Pour évaluer le modèle appris, nous avons cherché à prédire chaque variable au prochain pas de temps sur une observation ne faisant pas partie du jeu d'apprentissage. Nous avons répété ces opérations pour chaque observation. La prédiction est faite grâce à une inférence simple, partant de l'état courant cru approximé sous forme de filtre particulaire et chaque particule est inférée. Nous marginalisons alors le filtre particulaire, et la particule la plus probable devient la prédiction.

Dans un premier temps, nous sommes partis d'un modèle aléatoire pour l'apprentissage quantitatif (rappelons que EM est une recherche *locale*) ; nous avons alors constaté que le modèle appris était moins bon qu'une simple prédiction de type "rien ne change" (appelé modèle conservatif, noté mc). Nous somme alors partis d'un DBN à très forte probabilité que rien ne change, et avons recommencé l'apprentissage (nous l'appelons DBN conservatif, noté dbn c.). Les résultats sont alors bien meilleurs que précédemment, mais restent comparables à une prédiction de type "rien ne change". Nous y reviendrons plus bas. Les résultats sont résumés sur la figure III.2.

variable	hasard	dbn aléatoire	mc	dbn c.	mc (1/3)	dbn c. (1/3)
encombrement	33.3	74.3	88.4	83.9	50.3	65.3
angle de l'obstacle	25.0	70.1	87.4	88.7	54.4	70.2
dist. à l'obstacle	25.0	73.1	89.6	88.6	55.9	71.7
# segments	33.3	65.4	82.9	72.7	44.7	53.6
# vallées	33.3	49.8	80.2	74.8	56.8	65.8
v	33.3	59.4	90.3	82.6	49.1	60.8
ω	25.0	35.9	79.1	79.6	46.5	60.2
δ v	33.3	89.0	86.4	86.2	56.1	83.1
$\delta\omega$	33.3	90.2	86.5	89.1	69.3	88.9
δ dist. au but	33.3	59.7	71.5	69.4.8	48.3	62.3
% mission	33.3	44.4	96.5	95.9	78.0	84.3
stratégie	20.0	53.0	84.8	85.9	61.0	70.0
général	25.0	95.6	95.4	97.5	88.9	98.3
adéquation	33.3	49.3	92.1	93.5	67.0	77.4
moyenne	29.0	64.6	86.6	87.6	59.0	72.3

TAB. III.2: Tableau des prédictions correctes à un coup à fréquence donnée (%) (le 1/3 signifie que l'on a divisé la période par 3)

Une première conclusion est que partir d'un point de départ aléatoire pour l'apprentissage est une très mauvaise idée. Il est clair au vu de ces résultats que l'apprentissage s'arrête sur un optimum local très loin de l'optimum global. Cette considération ne doit surtout pas être oubliée lors de l'apprentissage automatique d'un modèle par l'algorithme EM.

D'autre part, on peut nettement voir avec ces résultats que pour nos expérimentations, un modèle conservatif simple est très bon, et qu'il semble inutile d'avoir recours à une modélisation stochastique complexe. Ceci est dû aux aspects dynamiques qui ont été délaissés dans un premier temps. En effet, aux vitesses auxquelles évolue le robot, les changements des variables apparaissent à une fréquence très faible. La fréquence de collecte des observations a été réglée de façon à être légèrement plus basse que la fréquence de fonctionnement du module de génération de mouvement ND, alors qu'elle aurait du être plus basse que la fréquence de changements de l'environnement. Mais nous n'avions aucun moyen de connaître cette fréquence a priori. Nous avons alors effectué une analyse des données simples pour faire la moyenne du nombre de fois qu'une variable ne change pas, et ce pour chaque variable. Nous avons constaté que la plus petite de ces moyenne est de 3, il semble donc que la dynamique de notre système soit a peu près trois fois plus lente que la période à laquelle nous avons collecté les observations. Nous avons alors appris un nouveau DBN pour cette fréquence. Cependant, toutes les variables n'ont pas la même fréquence de changement, et il faudrait utiliser cette information pour utiliser une structure de causalité plus fine pour le modèle.

Une autre utilisation de ce type de modèle peut être asynchrone. En effet, on peut considérer que les observations ne sont pas faites à fréquence constante, mais uniquement lorsqu'au moins une variable change. En effet, lors de l'utilisation, on peut décider de ne pas tenir compte des non-changements, car alors la décision n'aura pas à changer. Si l'on utilise cet asynchronisme, en considérant une nouvelle observation uniquement lorsqu'un certain nombre de variables variables change, on obtient le tableau III.3.

variable	mc (1)	dbn (1)	mc (2)	dbn (2)	mc (3)	dbn c. (3)
encombrement	81.5	80.8	76.9	75.6	71.6	68.7
angle de l'obstacle	86.7	86.4	83.1	82.5	78.2	77.0
dist. à l'obstacle	87.0	86.9	83.4	82.9	78.5	76.4
# segments	69.1	67.6	64.5	62.4	60.3	56.9
# vallées	70.0	70.1	65.7	65.1	62.3	60.6
v	79.7	79.3	73.8	72.9	67.1	62.4
ω	76.0	75.7	69.7	69.5	62.8	60.5
δ v	75.0	83.2	68.4	78.5	61.2	69.8
$\delta\omega$	78.3	86.6	72.5	83.4	66.9	79.2
δ dist. au but	57.2	65.9	51.2	62.5	48.1	54.1
% mission	95.3	95.1	94.0	93.8	92.1	91.0
stratégie	83.5	83.2	79.4	79.3	74.2	73.3
général	96.3	97.6	95.7	97.6	95.1	98.1
adéquation	92.3	92.3	90.0	89.9	87.2	86.3
moyenne	80.6	82.2	76.3	77.6	71.8	72.5

TAB. III.3: Prédictions correctes à un coup asynchrones (entre parenthèses : le nombre de variables minimum qui doivent changer pour qu'une nouvelle observation soit prise en compte)

Dans ce cas, on voit que les bénéfices par rapport à un modèle conservatif sont marginaux, et souvent pire pour ce qui est de prédire l'environnement. Il est clair que pour ce qui concerne l'environnement, il manque des variables pour mieux en capturer la dynamique. Cette approche événementielle n'est au finale pas meilleure que l'autre. Nous proposerons en fin de chapitre IV une idée pour mieux capturer les différences de dynamiques entre différentes composantes d'un même système. Nous pensons également que nos variables caractéristiques de l'environnement sont insuffisantes, et qu'étudier de nouvelles variables caractéristiques de la dynamique, de la présence d'humain autour du robot (obstacles mouvants) permettrait de construire de meilleurs modèles. De telles variables d'environnement plus fines ont été étudiées dans [Lampe et Chatila, 2006; Held, Lampe et Chatila, 2006] pour l'évaluation de performances de systèmes robotiques. Nous n'avons malheureusement pas eu l'occasion d'utiliser de telles variables.

Apprentissage structurel

Du coté de l'apprentissage structurel, nous n'avons pas implémenté de technique de recherche de BN optimal, ou d'ajout de variable cachée ou de lien causal. Nous sommes partis d'un DBN complet suivant notre structure, et nous avons plutôt cherché à éliminer des liens causaux en nous basant sur la notion d'information mutuelle (voir équations I.4).

Comme l'information mutuelle nous donne une mesure de l'indépendance de deux variables, on peut se servir de cette mesure comme heuristique de suppression d'un lien causal entre deux variables. En effet, si deux variables autour d'un lien causal ont une faible information mutuelle, il est probable que ce lien causal n'ait pas de raison d'être car les variables sont faiblement corrélées. On peut alors marginaliser les probabilités conditionnelles de la variable fille pour supprimer la variable parente à faible information mutuelle de la liste des parents. L'évaluation des informations mutuelles est très rapide, de l'ordre de la seconde pour chaque lien causal.

Il semble clair qu'il faut d'abord commencer par un apprentissage quantitatif jusqu'à atteindre un optimum de façon à ce que les informations mutuelles aient le plus de sens possible. On peut ensuite s'intéresser au retrait de lien causal. On remarque alors qu'en enlevant un lien causal, le DBN améliore son log-likelihood, ceci sans faire d'apprentissage quantitatif. À partir de là, plusieurs stratégies sont possibles pour apprendre un bon DBN en alternant phases de retrait de lien causal et phases d'apprentissage quantitatif. Soit on décide d'enlever un seul lien causal, puis de refaire un apprentissage quantitatif jusqu'à stabilisation, afin que les informations mutuelles aient le plus de sens possible avant la sélection du lien causal à enlever, soit on effectue plusieurs retraits de liens causaux jusqu'à ce que la probabilité de faire l'observation n'augmente plus, avant de repasser à l'apprentissage quantitatif.

Une autre considération est que lorsque l'on enlève un seul lien causal, la vraisemblance (likelihood) peut ne pas augmenter, mais si l'on enlève plusieurs liens causaux elle peut se mettre à augmenter. Autrement dit, cela signifie que le retrait de lien causal basé sur l'information mutuelle la plus faible n'entraîne pas une augmentation monotone de la probabilité de faire l'observation. Il faut alors déterminer jusqu'à quelle profondeur on essaie d'enlever des liens causaux, et mémoriser le meilleur DBN au cours de la recherche pour ne garder que celui-ci. Finalement, il ressort que même en se limitant au retrait de liens causaux, la recherche de bonne structure pour le modèle n'est pas une question simple.

Dans le cas du DBN synchrone (1/3), le log-likelihood correspondant au score de prédiction du tableau est de -23877, avant tout retrait de lien causal, et l'apprentissage structurel n'arrive plus à augmenter ce score. (Rappellons que le log-likelihood ll est tel que $P(O|\lambda) = e^{ll}$, avec λ le DBN courant). L'algorithme sélectionne ensuite les liens causaux par ordre d'information mutuelle, et sans même refaire plus d'apprentissage structurel, le likelihood augmente pour les premiers retraits de liens causaux.

lien causal enlevé	log-likelihood
aucun	-23877
variable cachée 1 → général	-23087
vitesse linéair<e maximale → variable cachée 2	-23040
élargissement du robot → variable cachée 1	-22969
vitesse linéaire maximale → variable cachée 0	-23044
... (à partir d'ici, le log-likelihood n'augmente plus)	

TAB. III.4: Retrait de liens causaux

Un premier effet important est que simplement en retirant ces trois liens causaux, la taille nécessaire pour stocker le DBN sur disque passe de plus de 600 Mo à moins de 250 Mo. L'empreinte mémoire est plus faible, mais est divisée suivant la même proportion. Ces considérations sont extrêmement importantes d'un point de vue applicatif, où l'idée est d'embarquer la gestion du DBN sur un robot, qui a besoin de mémoire pour beaucoup d'autres activités.

Nous avons également utilisé cette étude pour supprimer purement et simplement la variable de complexité géométrique. En effet, cette variable était très peu prévisible, et lors du retrait des liens causaux, tous les liens autour de cette variable étaient supprimé en premier, la rendant indépendante du reste du modèle. Notre avis sur ce fait est que dans notre jeu d'expérimentation cette variable variait très peu, et nous ne pouvions modéliser que du bruit. En effet, cette mesure a été utilisée pour différencier entre des environnement encombrés et des couloirs à structure simple, or nous n'avons jamais collecté de données dans un couloir.

Au niveau prédictif, on peut voir sur le tableau III.5 que les prédictions ne se sont quasiment pas dégradées, et parfois même localement améliorées.

Ces constatations nous amènent à la conclusion que la structure et les variables utilisées dans cet exemple sont sous-optimales. En effet, il semble manifeste que les résultats ne devraient pas changer, ou très peu. Que retirer un lien causal permette de faire de meilleures prédictions est un signe manifeste de problèmes de précision en machine. D'autre part, il est clair que le nombre de variables cachées ainsi que leur arité ne permet pas de supprimer simplement des liens causaux et de trouver la "vraie" structure de causalité du processus modélisé. En utilisant plus de variables cachées de plus grande arité, les résultats sont nettement meilleurs, au prix de temps d'apprentissage et de taille mémoire du DBN rédhibitoires.

Un autre effet que nous aurions souhaité aurait été de pouvoir discerner plusieurs sous-système faiblement couplés dans le modèle, mais avec si peu de variables cachées, cela n'est pas possible.

variable	dbn avec tous liens causaux	dbn après retraits
encombrement	68.7	66.1
angle de l'obstacle	77.0	70.3
dist. à l'obstacle	76.4	71.5
# segments	56.9	53.7
# vallées	60.6	63.6
v	62.4	60.2
ω	60.5	58.8
δ v	69.8	82.3
$\delta\omega$	79.2	88.6
δ dist. au but	54.1	61.7
% mission	91.0	84.8
stratégie	73.3	70.0
général	98.1	98.3
adéquation	86.3	77.3
moyenne	72.5	71.9

TAB. III.5: Comparaison avant et après le retrait de liens causaux non informatifs

Il pourrait alors être intéressant d'appliquer le type d'algorithme qui a été utilisé pour déterminer un bon nombre d'états cachés pour un HMM, dans l'idée de mélanger finement connaissance humaine et apprentissage automatique pour la structure et l'arité des variables. Nous n'avons malheureusement pas pu approfondir dans cette direction.

Chapitre IV

Applications des modèles graphiques

Dans ce chapitre, nous allons détailler plusieurs applications autre que la modélisation et le contrôle d'un robot. En effet, ce type de modèle est extrêmement générique, et peut être utilisé hors de la robotique. Nous cherchons ici à illustrer cette généricité, à travers diverses utilisations beaucoup moins liées à la robotique que notre exemple récurrent de navigation autonome.

Dans un premier temps, nous allons nous pencher sur des considérations générales et des applications déjà existantes, puis allons voir des exemples sur lesquels nous avons expérimenté nous-mêmes, dans le cadre de la reconnaissance de geste, mais aussi pour la reconnaissance de processus mentaux.

IV.1 Types de Structures

Comme nous l'avons déjà souligné, toutes sortes de structures peuvent être représentées par des réseaux bayésiens dynamiques. Leur seule limite est une stationnarité des liens causaux, c'est-à-dire que la structure elle-même ne varie pas au cours du temps, et que les probabilités conditionnelles sont indépendantes du temps. On utilise en général des DBN répondant à l'hypothèse Markovienne d'ordre 1, car on peut contourner cette limitation en ajoutant des variables cachées ou en augmentant leur arité, de façon à prendre en compte un certain niveau d'historique. Une variable peut ainsi être ajoutée pour ramener au pas de temps $t-1$ une valeur d'une autre variable antérieure, par exemple.

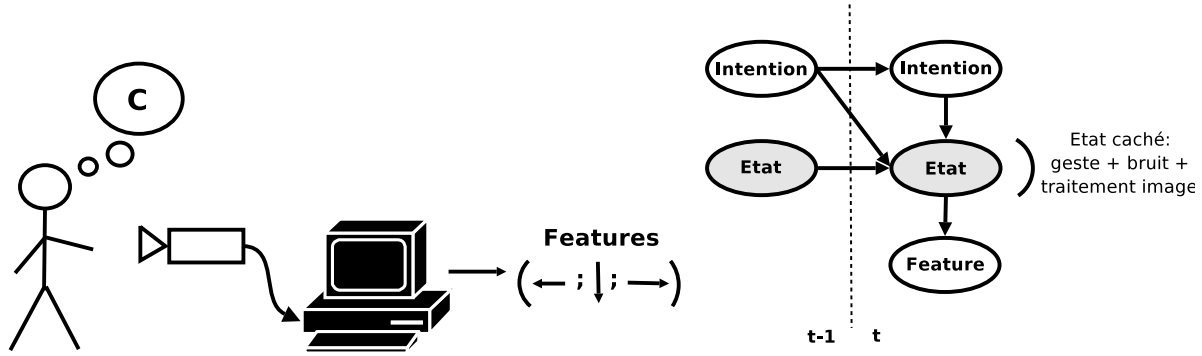
Sens de la Causalité

Dans un réseau bayésien, le sens des flèches représentant les liens causaux est souvent interprété intuitivement comme la causalité au sens commun du terme, or c'est un abus par rapport à la définition. En effet, le sens de la flèche indique simplement quelle est la probabilité conditionnelle qui est stockée dans les données quantitatives attachées au lien. Et il n'y a pas de causalité dans une probabilité conditionnelle, on peut très bien retourner ces probabilités en utilisant la règle de Bayes. Dans certains articles, on peut trouver une nuance entre réseau bayésien "purs", c'est-à-dire dans lequel le sens des flèches n'apporte aucune information sémantique, et réseau

bayésiens causaux, dans lesquels les liens causaux sont en plus interprétés avec une sémantique de causalité intuitive.

Dans le cadre des réseaux bayésiens dynamiques, comme on ajoute une donnée temporelle, il est clair qu'une variable ne peut avoir d'influence sur le passé, et il serait à la fois contre-intuitif et contre-productif de stocker les probabilités conditionnelles sur des liens allant de t vers $t-1$. Néanmoins, même dans le cadre où l'on impose que le sens des liens causaux soit celui de la causalité, plusieurs possibilités restent offertes pour modéliser un même système.

Prenons comme exemple un système composé d'une personne faisant un geste devant une caméra, et processus de traitement de l'image correspondant, représenté sur la figure IV.1(a). Le but de cet exemple est de modéliser l'ensemble du dispositif de reconnaissance, en vue éventuellement d'automatiser la reconnaissance du geste (mais nous ne dirons pas ici comment aller jusqu'au bout de la reconnaissance, nous y reviendrons dans la section IV.1).



(a) Vue schématique (b) DBN correspondant

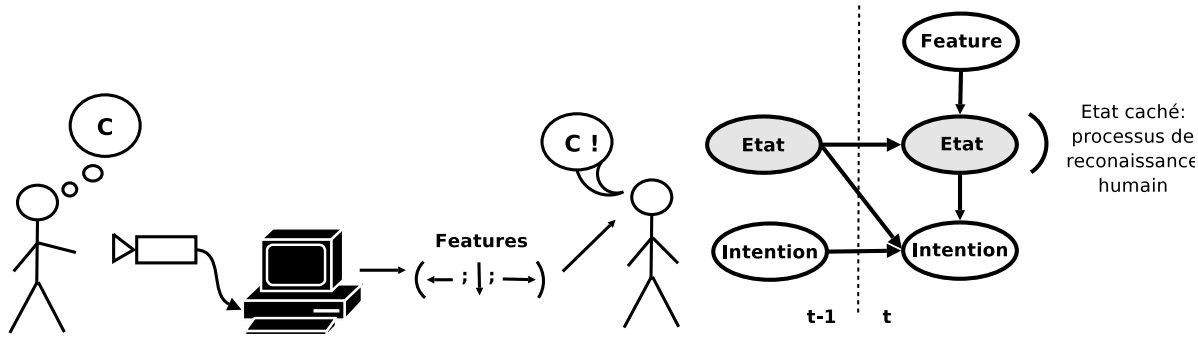
FIG. IV.1: Exemple de système construit en vue de reconnaître un geste

Une première approche serait de représenter ce système par un DBN complet, par exemple par celui représenté sur la figure IV.1(b), où le DBN est un modèle du système opérateur/traitement de l'information.

Néanmoins, un point de vue quasiment équivalent peut être présenté sur la figure IV.2(a), qui représente le système complet qui nous intéresse : le processus qui produit le signal, et l'observateur qui le reconnaît.

Et, du coup, si l'on veut effectivement reconnaître le geste, il peut être plus intéressant de modéliser celui qui reconnaît que le système lui-même. Le DBN correspondant peut être vu sur la figure IV.2(b).

On constate alors que les DBN sont purement et simplement inversés au niveau du sens des liens causaux. Dans un cas, la causalité est celle de la création du geste, de l'opérateur vers le traitement du signal, alors que dans l'autre cas le signal crée le geste dans l'esprit d'un deuxième opérateur qui doit le reconnaître (et alors la variable *intention* est ce que le deuxième opérateur déduit de l'intention du premier). De plus, ces deux approches sont équivalentes au niveau sémantique car elles permettent toutes deux de faire la reconnaissance. Pour une telle application de reconnaissance de geste, le seul critère qui reste pour départager ces deux approches, est uniquement de type calculatoire. En fonction de l'arité des variables et des liens causaux , il faut



(a) Vue schématique (b) DBN correspondant à l'observateur

FIG. IV.2: Mise en évidence de l'observateur

choisir la structure qui sera la plus légère autant pour l'apprentissage que pour la reconnaissance elle-même, même si celle-ci est normalement bien moins coûteuse que l'apprentissage.

A travers cet exemple, on peut constater que parfois plusieurs structures proches mais inversées causalement sont possibles. Cette remarque est générale et doit être gardée à l'esprit lorsque l'on veut modéliser un système sous forme de DBN ; il faut alors prendre en compte la complexité d'apprentissage qui peut fortement varier d'un modèle à l'autre.

Reconnaissance

Pour faire de la reconnaissance de processus, le principe est le suivant : si l'on reprend notre exemple de reconnaissance de gestes automatique via une caméra, on peut modéliser pour chaque geste les informations du processus par un HMM.

Étant donnés une séquence d'observations pour le geste à reconnaître, et un modèle appris pour chaque geste possible, la reconnaissance consiste à calculer les probabilités de voir la séquence à reconnaître suivant chaque modèle. Alors le meilleur modèle sera celui du geste d'entrée le plus probable. Le principe est illustré figure IV.3.

Hiérarchie

Cependant, avec ce genre d'approche, on est obligé de construire plusieurs modèles différents, et si certaines informations sont communes entre plusieurs modèles, on est obligé de les dupliquer. Une façon de voir la hiérarchie est utilisée dans la section IV.2 : il s'agit d'un façon de grouper des modèles pour différents processus, mais au sein d'un même modèle ayant une variable "de haut niveau" qui permet de différencier chaque processus à reconnaître.

On peut alors penser à ajouter à ces HMM une variable hiérarchiquement supérieure, encore une fois au niveau sémantique. Cette variable sera alors le type de modèle, ou encore le nom du geste dans notre exemple. On aura alors un DBN de la forme de la figure IV.4(a). Lors de la phase d'apprentissage, cette variable haute sera observable, et ensuite on la considérera comme cachée pour la reconnaissance elle-même, et on cherchera à inférer sa valeur.

Pour inférer sa valeur, deux cas se présentent alors :

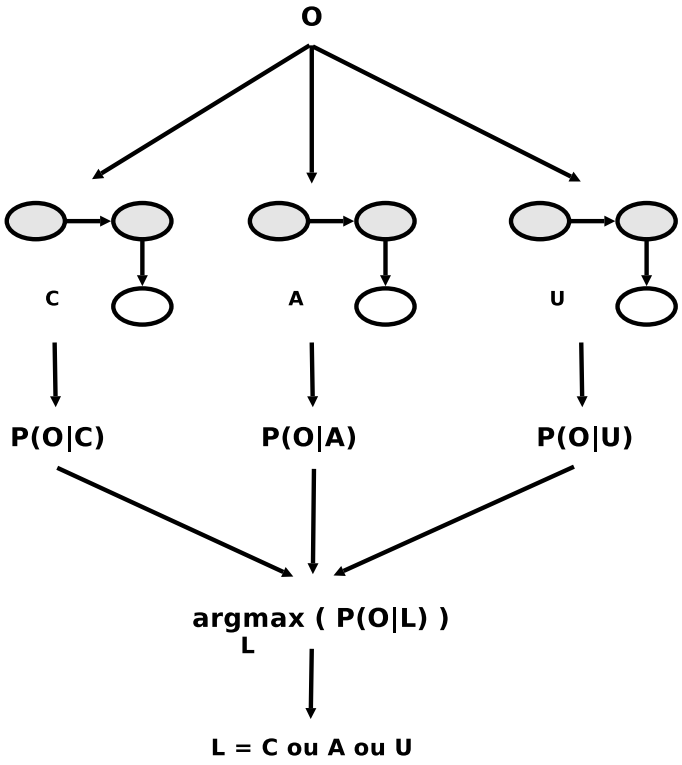
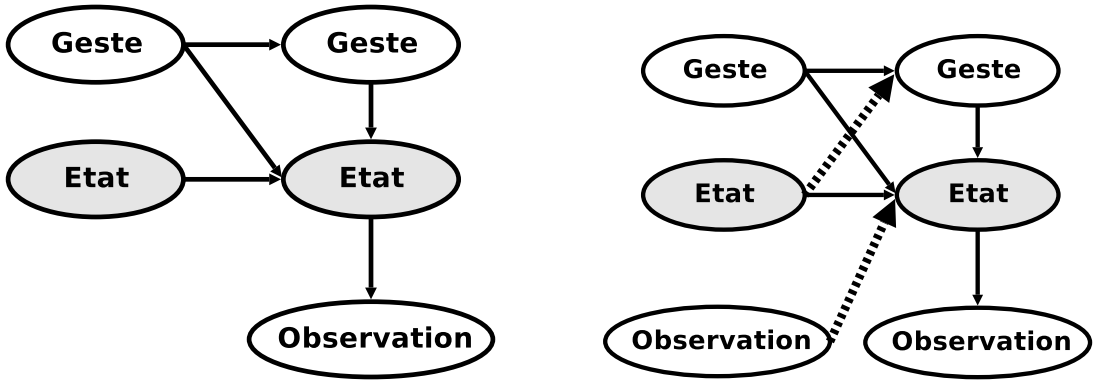


FIG. IV.3: Principe de reconnaissance de processus basée sur des HMMs

– si l'on fait une inférence mono-hypothèse (de type exact par exemple), il faut alors pouvoir calculer la valeur de cette variable qui est devenue cachée à partir des liens causaux présents dans le DBN. Pour cela, il faut ajouter un lien qui n'est pas intuitif uniquement pour pouvoir faire les calculs. De fait, il est un lien causal de l'observateur du système. Ce lien causal est mis en évidence dans la figure IV.4(b).
– En revanche, si l'on fait une inférence multi-hypothèse (type filtrage particulaire) les particules (les hypothèses) correspondantes aux mauvaises valeurs de cette variable (i.e. aux gestes qui ne sont pas celui à reconnaître) sont éliminées au fur et à mesure de l'avancée de la reconnaissance car elles n'expliquent pas bien l'observation. Dans ce cas, ce lien causal supplémentaire n'est pas obligatoire.

Ainsi, il faut relativiser l'intuition de la causalité dans un réseau bayésien. Les liens dits "causaux" ne représentent au final que des probabilités conditionnelles, et mettre sur ces probabilités une sémantique de causalité n'est pas toujours intuitif. Dans le cas de la reconnaissance de gestes, on peut dégager une causalité du système lui-même et une de l'observateur du système, et deux structures opposées permettent les mêmes utilisations. Finalement, on peut mettre les deux types de liens causaux dans le même modèle graphique, comme nous allons le voir plus bas.

Dans tous les cas, on peut constater que l'on peut calculer les valeurs de la variable à deviner avant même la fin de la séquence d'observations. Ceci est également valable pour le cas ou l'on utilise plusieurs modèles, en calculant $P(O_{1:t}|\lambda_i)$, avec $O_{1:t} \subset O$.



(a) DBN simple

(b) Liens causaux observateur ajoutés pour simplifier l'inférence

FIG. IV.4: Modèle DBN hiérarchique pour la reconnaissance de processus

Des structures plus complexes ont été proposées pour des applications particulières. Si au niveau uniquement formel les auteurs de [Liao, Fox et Kautz, 2004] restent dans le strict cadre DBN, leur approche peut être qualifiée de hiérarchique dans le sens où les variables ont une sémantique hiérarchique, avec les variables "en haut" (causes) de très haut niveau d'abstraction, et des variables "basses" (conséquence) de beaucoup plus bas niveau, avec plusieurs niveaux intermédiaires. On peut voir dans leur approche que les liens causaux ne sont alors plus forcément des liens de causalité au sens intuitif du terme, et qu'ils mélangent des causalités propres au système et d'autres propres à l'observateur, ceci en vue de faire des déductions plus efficaces. Dans leur application, les auteurs cherchent à modéliser les déplacements d'un sujet dans une ville depuis son but jusqu'à sa position GPS, en passant par des segments de déplacement (aller d'un quartier à un autre en bus, par exemple) et des modes de déplacement (pied, bus...). On constate alors que, dans leur modèle, il y a un lien causal du but du sujet vers son segment de déplacement, ce qui implique une logique propre au sujet : "aller à la banque" implique de "prendre le bus jusqu'au quartier correspondant", par exemple. Mais on peut aussi voir des liens causaux de sens inverse, où une variable de segment de déplacement influe sur une variable de changement de but qui influe sur le but du sujet. Ceci est une logique propre à l'observateur, et est un non-sens si l'on ne considère que le sujet.

Ces travaux se basent sur un modèle plus général détaillé dans [Bui, 2003; Bui, Venkatesh et West, 2002] qui propose une méthode élégante pour reconnaître des comportements ou des politiques de haut niveau, à travers des modèles avec ou sans mémoire (la mémoire est alors simplement une variable supplémentaire dans le DBN). Cette méthode est très générale, mais le problème est alors de construire ou d'apprendre un tel modèle, la reconnaissance étant alors simple via une méthode à base de filtre particulaire. Dans ces travaux, les auteurs apprennent le DBN général par tranches d'abstraction séparément (voir [Liao, Fox et Kautz, 2005], puis "collent" ensemble les morceaux pour obtenir le DBN global. L'idée fondamentale est alors d'apprendre simultanément un modèle du système et un modèle pour la reconnaissance, tel que nous le faisons dans l'application à la reconnaissance de gestes.

IV.2 Reconnaissance de gestes

Nous avons pu tester la reconnaissance de gestes grâce à ce type de modèle. Les modèles de Markov cachés ont déjà été utilisés pour la reconnaissance de geste, et ont prouvé leur efficacité [Yoon et al., 2001]. Cependant, nous pensons qu'utiliser des DBN sous la forme hiérarchique comme détaillée plus haut peut s'avérer plus efficace encore.

Un aspect particulièrement important pour ce type d'application est l'aspect séquentiel de la reconnaissance. En effet, on dispose de beaucoup d'information sur ce que doit être le modèle, et en particulier, on sait que la reconnaissance doit avoir un aspect *séquentiel* ; c'est-à-dire qu'une partie de l'état caché ne boucle pas. Ce type d'approche est connu pour les HMMs et constitue les modèle dits "left-right" [Rabiner, 1989], c'est-à-dire que l'état caché ne peut qu'avancer dans ses valeurs, et que le processus ne contient pas de boucles. C'est exactement le même cas dans la reconnaissance de gestes. Si on est en train d'observer un mouvement horizontal pour reconnaître un "C", par exemple, alors il est intéressant de pouvoir séparer les deux occurrences, car on sait que si elles sont séparées par un mouvement horizontal, alors elles sont différentes. Le processus modélisé ne contient pas de boucle, et le deuxième état décrit doit être différent car séparé temporellement.

Dans le cadre DBN, il est encore plus facile d'intégrer cet aspect, simplement en rajoutant une variable cachée qui a des probabilités de transitions avant apprentissage qui reflètent ce comportement, c'est-à-dire que les probabilités de ne pas changer de valeur ou de passer à la valeur suivante sont démesurément grandes par rapport aux autres.

Le schéma devient alors celui présenté sur la figure IV.5, où l'on rajoute simplement une partie à l'état caché qui devient "left-right", c'est-à-dire qui ne peut qu'augmenter ou rester à la même valeur.

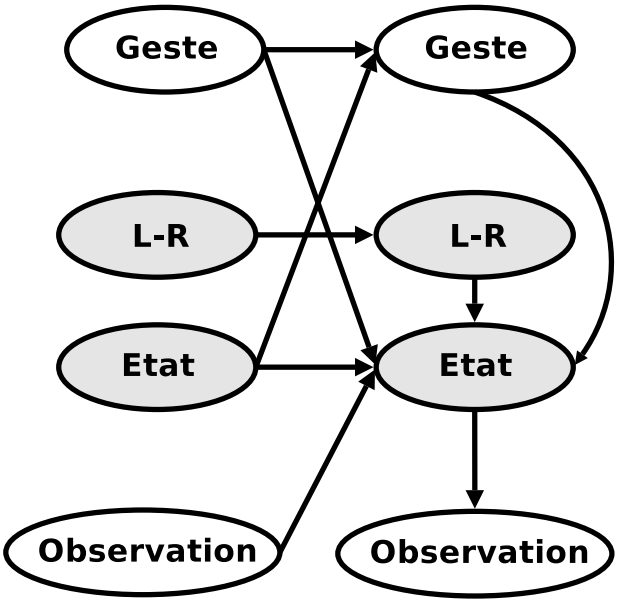


FIG. IV.5: Ajout d'une partie orientée à l'état caché

Nous avons expérimenté la reconnaissance avec différentes structures sur des données issues de l'utilisation de modèles dynamiques pour le suivi des gestes expliqués dans [Brèthes, Lerasle et Danès, 2005] et [Brèthes, 2005], chapitre 5. La contrainte était de n'utiliser que le modèle dynamique utilisé pour améliorer le suivi parmi *horizontal*, *vertical* et *arrêt*, afin de voir si cette information est suffisante. Les gestes qui ont servi de données sont constitués par quatre lettres "C", "L", "O" et "U" dessinées en l'air par un opérateur devant une caméra.

Le type de modèle que nous avons appris est asynchrone, de type événementiel, c'est-à-dire que nous partons des données synchrones et filtrons pour qu'une nouvelle observation ne soit générée que s'il y a un changement par rapport à la précédente.

Les différents modèles graphiques utilisés sont constitués par :

1. une reconnaissance classique à base de 4 HMMs, un par modèle, et donc avec une seule variable état ;
2. une reconnaissance via 4 DBNs à 1 variable état et une variable L-R (non hiérarchique donc, mais avec deux variables cachées en tout) ;
3. une reconnaissance par modèle hiérarchique sans variable L-R ;
4. et enfin une reconnaissance par modèle hiérarchique avec variable L-R.

Le nombre de variable cachée est toujours de 1. Il est à noter que si on augmente le nombre de variables cachées, la reconnaissance est meilleure au prix de temps d'apprentissages plus longs. L'arité de la variable L-R est de 10 ; dans les cas où on n'utilise pas de variable L-R, on utilise une variable cachée d'arité 10, et sinon 5 (pour avoir des temps d'apprentissage très courts, de l'ordre de dizaines de secondes). Encore une fois, augmenter ces arités permettrait d'obtenir de meilleurs scores, au prix de temps d'apprentissages beaucoup plus longs.

Les résultats de reconnaissances correctes sont donnés sur le tableau IV.1. Dans le cas hiérarchique, comme nous déduisons le geste reconnu à chaque pas de temps via le filtre particulaire, nous disposons de plusieurs façons de déduire le geste correspondant. On peut donner le geste déduit au dernier pas de temps (noté valeur finale), le nombre de fois que la valeur de cette variable est bonne globalement (noté score global), ou encore à la fin de chaque observation regarder quel est la valeur majoritaire de la variable *geste*, (noté vote). Pour la forme HMM, non hiérarchique, on déduit simplement quel est le modèle qui explique le mieux le geste à la fin de chaque séquence de test.

	forme hiérarchique				forme HMM	
	modèle	score global	vote	valeur finale	modèle	score
avec variable L-R	(4)	45.7	50.0	67.3	(2)	64.5
sans variable L-R	(3)	43.1	49.0	60.6	(1)	59.6

TAB. IV.1: Taux de reconnaissance suivant les structures (%) (entre parenthèses, le numéro du modèle tel que décrit plus haut)

En augmentant l'arité des variables (25 pour le cas sans variable L-R et 15 avec), on obtient les résultats du tableau IV.2, qui sont nettement meilleurs au prix d'un apprentissage nettement plus long.

	forme hiérarchique				forme HMM	
	modèle	score global	vote	valeur finale	modèle	score
avec variable L-R	(4)	45.1	53.8	67.3	(2)	65.4
sans variable L-R	(3)	44.1	48.1	67.3	(1)	66.3

TAB. IV.2: Taux de reconnaissance suivant les structures (%) (arité de 25, 10 et 15 pour une seule variable, la variable L-R et la variable non L-R respectivement)

On constate que sans variable L-R (modèle 3), le modèle hiérarchique se comporte de la même façon, voire un tout petit peu mieux que le vote des différents HMMs, pour peu que l'on tienne compte de toute l'observation. Ceci est normal, car il est clair que le DBN ne peut déduire dès les premiers instants quel est le geste à reconnaître. Il a en théorie besoin de toute la séquence. L'équivalence des résultats est due au fait que, dans ce test, nous n'avons utilisé qu'une seule variable cachée de même arité que pour le HMM, et donc la forme repliée factorisée est à peu près équivalente. Nous pensons que le fait qu'elle soit très légèrement meilleure est dû à plus de données d'entraînement, car ce DBN a vu toutes les données, contrairement aux 4 HMMs qui voient chacun un quart des données d'entraînement. Pour peu qu'il y ait un minimum de redondance dans ces données, la structure hiérarchique est à même d'en tirer parti, contrairement au modèle simple.

On constate également que la variable L-R permet de bien mieux guider l'apprentissage et donne de bien meilleurs résultats.

IV.3 Brain-Computer Interfaces

Une autre application de choix, toujours dans le domaine de la reconnaissance de processus se situe au niveau des interfaces cerveau/ordinateur. Ce domaine connaît un intérêt croissant de part de nombreuses applications en rapport avec des applications médicales, pour la commande de prothèses ou d'aides diverses.

Dans [Rebsamen et al., 2006], on peut voir le contrôle d'un fauteuil roulant basé sur la détection de signaux à partir d'électro-encéphalogrammes (EEG) d'une personne enregistrés à partir de casques standardisés à 64 points de mesures (voir figure IV.6). Ces signaux sont ensuite décodés pour donner des ordres de destination à un fauteuil roulant entièrement robotisé.

Plusieurs approches ont été utilisées pour obtenir des informations sur l'état mental d'une personne de manière non-invasive. Toutes se basent sur l'acquisition d'EEG et sur une phase d'apprentissage, mais sont très différentes.

Dans [Millán, Renkens, Mouriño et Gerstner, 2004], les auteurs demandent à l'opérateur de se concentrer intensément sur différents états mentaux (qui peuvent être de penser à une certaine couleur, ou penser à bouger telle ou telle partie du corps), et après une phase de pré-traitement, un apprentissage automatique essaie de séparer ces différents états mentaux à l'intérieur même des EEG. Dans cet article, un classifieur gaussien est appris sur le modèle de l'algorithme décrit dans [Bishop, 1995], mais d'autres techniques sont également possibles [Hauser, Sottas et Millán,

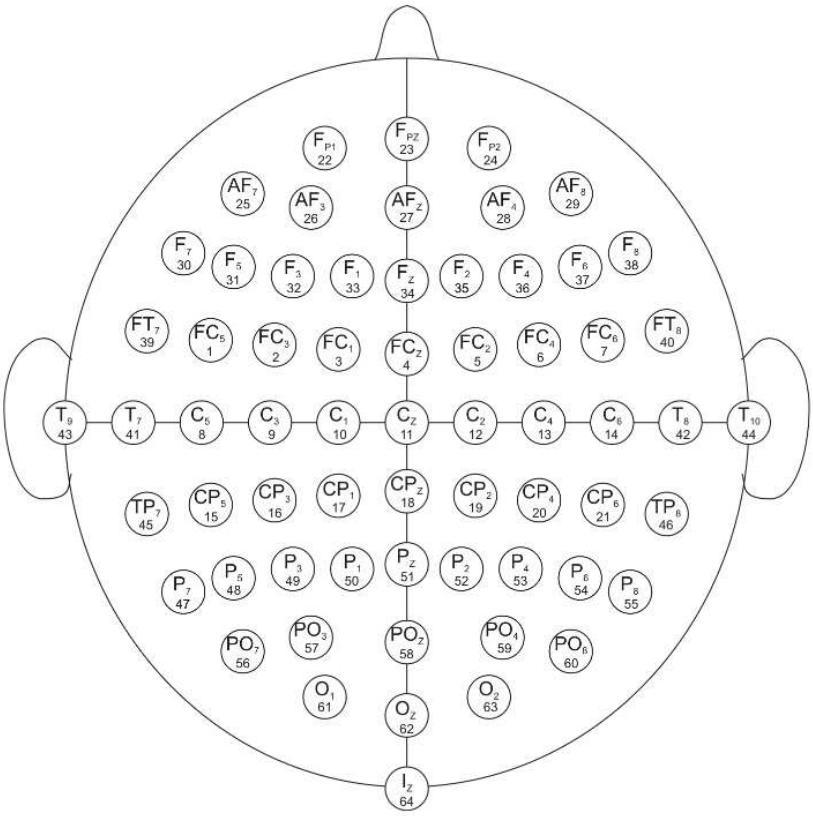


FIG. IV.6: Schéma des points de mesures standard d'un EEG

2002]. Le gros défaut de cette approche est que l'opérateur lui-même doit passer beaucoup de temps à apprendre à bien séparer des états mentaux, et surtout à se concentrer intensément dessus.

Une autre approche plus légère pour l'opérateur se base sur la détection d'un signal particulier dans l'EEG nommé *"P300"*. Il apparaît dans le cadre de recherches récentes en neurosciences que la prise de conscience d'un stimulus visuel attendu parmi du bruit se fait à peu près 300 ms secondes après l'apparition de ce stimulus. L'activité de certaines zones du cerveau change entièrement selon que la prise de conscience de ce stimulus se fait ou non. Ceci se traduit par l'apparition du signal *"P300"*. L'idée utilisée pour le contrôle du fauteuil roulant est alors de placer un écran devant l'opérateur qui affiche différentes cases où sont écrites des destinations possibles. Ces cases clignotent aléatoirement, et l'opérateur doit simplement fixer la case où est écrite sa destination, pendant que son EEG est enregistré. Ici encore, la phase d'apprentissage du classifieur est une étape critique, mais l'opérateur n'a pas d'effort mental à soutenir. Pour la détection d'un tel type de signal, sont en général appris des classifieurs à base de Support Vector Machines (SVM) [Vapnik, 1998; Cristianini et Shawe-Taylor, 2000]. Ces techniques rendent implicite la phase d'extraction des données caractéristiques grâce à l'utilisation d'une fonction noyau appliquée directement sur les données brutes. Différentes fonctions noyaux sont connues pour très bien marcher dans la plupart des cas.

Le défaut de ce type d'approche est de ne pas considérer l'aspect dynamique qui peut être nécessaire pour des applications telles que la reconnaissance de processus mentaux à partir de l'activation séquentielle de différentes parties du cerveau. Nous avons donc essayé d'apprendre un réseau bayésien dynamique sur des données EEG.

Pour cela, nous avons utilisé un des jeux de données de la *BCI Competition* [*] , proposé par les auteurs de [Millán et al., 2004]. En effet, ce jeu de données est le seul pré-traité, alors que les autres sont des EEG bruts, sans que nous sachions quels sont les types de traitements, en fréquence en particulier, à y appliquer. Cette compétition propose différents jeux de données d'apprentissage et des jeux de tests pour lesquels il faut reconnaître les états mentaux des sujets.

Dans ce jeu de données, l'opérateur doit se concentrer sur trois tâches mentales différentes parmi "imaginer bouger la main droite", "imaginer bouger la main gauche" et "imaginer des mots commençant par une certaine lettre". Les données collectées sont les EEG bruts à une fréquence de 512 Hertz. Ces EEG sont filtrés par un laplacien de surface puis à 16 Hertz sont calculés les densités de pouvoir spectral (PSD) sur des bandes de fréquences de 8 à 30 Hertz sur la dernière seconde pour les 8 canaux centraux-pariétaux (les 8 les plus centraux sur le crâne). Chaque donnée traitée est alors un vecteur de 96 dimensions (8 canaux pour 12 composantes fréquentielles) plus la classe de la tâche mentale pour les données d'apprentissage. Trois sujets différents ont été mis à contribution pour obtenir différents jeux de données. Le meilleur algorithme utilise une classification simple grâce à un discriminateur DB [Cuadras, Fortiana et Oliva, 1997] avec une distance euclidienne, mais les bons résultats sont surtout dus à la détection de changement d'activité mentale basée sur un seuillage des données, ce qui permet aux auteurs de ne pas choisir parmi les trois tâches, mais parmi les deux qui ne sont pas l'ancienne. Les autres algorithmes, aux résultats comparables, se basent tous sur l'utilisation de SVM, avec des pré-traitements des données ou augmentation des données par la sortie d'autres classifieurs.

Notre approche est très différente. Nous commençons par discrétiser les données traitées (les PSD) en utilisant un classifieur de kohonen non supervisé sans tenir compte de la tâche mentale en cours. Le nombre de PSD différents que nous avons utilisé est de l'ordre de la vingtaine. Nous construisons ensuite un DBN hiérarchique pour la reconnaissance, avec en variable haute la tâche mentale, et en observation les PSD discrétisés. Nous effectuons alors le même type de reconnaissance que pour les gestes.

Une première remarque est qu'avec un nombre d'états cachés possibles petit, l'algorithme n'apprend rien du tout. Ceci est logique du fait de la complexité de l'application. Apprendre un modèle avec moins de 10 états cachés différents ne permet pas d'apprendre des états discriminants. Nous avons utilisé 2 variables à 5 valeurs différentes, de façon à avoir des temps d'apprentissage relativement courts (quelques heures pour plus de 10 000 vecteurs de données d'apprentissage). Au vu de la dynamique du système, il est également clair que l'observation ne varie pas à chaque fois. Nous avons donc initialisé la table de probabilité d'une des variables cachées à une table "conservative", c'est-à-dire où la probabilité que la variable change de valeur est assez faible.

Les résultats sont alors les suivants[†] :

[*] http ://ida.first.fraunhofer.de/projects/bci/competition_iii

[†] voir http ://ida.first.fraunhofer.de/projects/bci/competition_iii/results/index.html#martigny

auteur	moyenne	sujet 1	sujet 2	sujet 3	notes
Ferran Galan	68.65	79.60	70.31	56.02	Algorithme dédié
Xiang Liao	68.50	78.08	71.66	55.73	SVM (sur données transformées)
Walter	65.90	77.85	66.36	53.44	SVM (données augmentées)
Notre algorithme	**47.69**	**60.87**	**44.44**	**37.76**	

TAB. IV.3: Résultats en reconnaissance correcte (%)

Les résultats sont extrêmement moyens par rapport aux meilleurs résultats que nous avons rapporté sur le tableau, mais l'approche semble intéressante. En effet, nous avons utilisé très peu de classes d'observation (de l'ordre de la vingtaine), et une seule variable cachée à 20 valeurs différentes. Le score augmentant avec le nombre, l'arité des variables et le nombre de classes d'observation, de bien meilleurs résultats auraient été obtenus en utilisant un plus grand modèle, nécessitant malheureusement trop de temps d'apprentissage pour les tests que nous avons pu effectuer (les jeux de données d'apprentissage et de test sont très larges). Mais le fait est que dans cette expérience, l'algorithme apprend effectivement quelque chose, même si le modèle obtenu reste imprécis. Les autres méthodes sont complètement spécifiques, et nous aurions également pu penser à ajouter une variable heuristique de détection de changement d'état mental comme dans le meilleur algorithme pour cette application.

IV.4 Remarques

L'aspect dynamique du processus à modéliser est extrêmement important pour construire un bon modèle. En effet, si les observations sont trop fréquentes par rapport à la dynamique du système, l'apprentissage du modèle va complètement noyer les transitions significatives dans le bruit dû aux non-changements. Plusieurs solutions sont alors à prévoir.

Approches classiques

Comme nous l'avons déjà dit, la phase de pré-traitement des données et d'ingénierie est fondamentale. Celle-ci ne devrait pas se passer d'analyse des aspects temporels du système. Il est nécessaire d'analyser finement la dynamique du système avant apprentissage, et au besoin baisser la fréquence des données utilisées. Cependant, il peut arriver que les différentes données utilisées n'aient pas la même dynamique, et alors le problème reste entier.

On peut aussi passer à un modèle asynchrone basé sur la détection de nouveaux événements. Si cette approche est très séduisante, elle rend impossible l'utilisation du modèle pour prédire l'évolution temporellement. En effet, le modèle va complètement changer de sémantique et permettra de dire que tel ou tel changement est probable dans un certain nombre d'étapes, mais la durée d'une étape est alors inconnue. Si cette approche est bonne pour faire de la reconnaissance après-coup, elle est inutilisable si l'on a besoin de faire de la prédiction temporelle, c'est-à-dire utiliser le modèle pour deviner dans combien de temps tel ou tel événement risque de se produire.

D'autre part, certaines approches ont vu le jour pour modéliser par un seul HMM différentes observations qui n'apparaissent pas à la même fréquence [Bengio, 2004, 2003]. L'idée est d'ajouter une variable cachée qui conditionne l'apparition de l'observation la moins fréquente par rapport à la plus fréquente (qui apparaît à chaque pas de temps). Cependant, si des algorithmes efficaces ont été proposés pour apprendre de tels modèles, l'hypothèse principale est que les observations sont conditionnées par le même état global, et ne permet de pas de dégager de sous-processus, s'il y en a, alors que les DBNs permettent cela.

Vers un DBN à synchronicités multiples

Nous proposons une troisième méthode inspirée de l'ajout de variable "L-R" dans la reconnaissance de gestes. Lors de l'apprentissage de modèle de navigation robotique, les différentes observations ne changent pas du tout à la même fréquence, et il devrait en être de même pour les variables cachées, de façon à mieux définir le comportement temporel du système.

Comme illustré sur la figure IV.7, nous proposons de découper les variables cachées en deux sous-variables, l'une muette dont on laisse à l'algorithme d'apprentissage le soin de définir entièrement les valeurs, et s qui permettra de capturer les effets synchrones. Pour ceci, on initialisera simplement ses probabilités de transition de façon à ce que ses valeurs avant apprentissage changent de façon périodique et cyclique, c'est-à-dire que cette variable ne pourra que rester à la même valeur, augmenter ou revenir à zéro quand elle aura atteint sa valeur maximale.

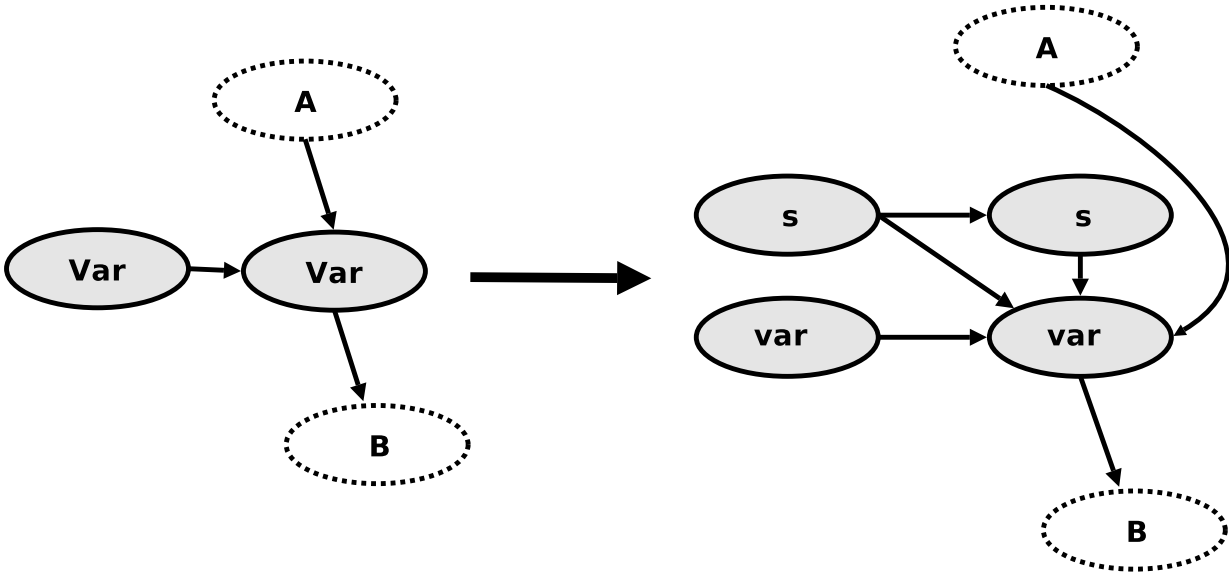


FIG. IV.7: Transformation d'une variable cachée classique en une variable cachée synchronisée par s

Le problème de cette approche est que, dans ce cas, on multiplie le nombre de variable cachées et que l'apprentissage en sera d'autant plus alourdi. Il faut donc limiter le nombre de liens

causaux influençant cette variable. C'est pour ça que nous proposons un minimum de liens causaux supplémentaires.

On peut alors imaginer décomposer entièrement le processus en un ensemble de variables avec composante de synchronisation, par exemple sur le modèle d'une décomposition en série de Fourier i.e. une variable L-R qui est de même période que le processus à modéliser (et qui donc ne revient pas à zéro), une de fréquence double, une de fréquence quadruple etc.

Après un premier apprentissage, on pourra alors voir sur le modèle d'apprentissage structurel quelles sont celles qui sont utiles, et enlever les autres. Cette méthode permettrait d'être sûr de ne rater aucune des composantes de fréquences différentes du processus. Cependant, il faut rester très prudent quant au nombre de ces variables, car du fait de la complexité du processus d'apprentissage, trop de variables pouvant prendre trop de valeurs différentes rend l'apprentissage impossible, tant au niveau du temps que de la mémoire nécessaires. Il peut alors être intéressant d'utiliser la phase d'ingénierie décrite comme première solution "classique" de façon à limiter le nombre de ces variables synchronisées.

Chapitre V

Prise de Décisions en Robotique

Comme nous l'avons déjà souligné, le type de modèle à base de réseaux bayésiens dynamiques que nous avons construit peut être utilisé tel quel pour la prise de décision de très haut niveau, en donnant par exemple les probabilités de succès, les niveaux de ressources attendus... En fait ce type de modèle prédictif nous permet de calculer les probabilités sur toutes les variables dont on dispose, sur n'importe quel horizon.

Nous allons dans ce chapitre mettre en avant une méthode simple et efficace pour utiliser ce type de modèle dans le cadre de l'optimisation en ligne du processus à modéliser, ceci en vue du contrôle fin de notre exemple de navigation robotique.

Ce chapitre présente principalement des prospectives basées sur une vision plus pragmatique que ce qui est présenté habituellement. Les résultats présentés sont préliminaires à une prise de décision complète telle qu'étudiée par ailleurs dans des cadres classiques de processus décisionnels de Markov, ou dans des cadres d'apprentissage par renforcement. L'originalité nécessaire à notre application vient du fait que le problème auquel nous nous intéressons est de taille beaucoup plus importante que les exemples habituels.

V.1 Réseau Décisionnel Dynamique

La façon la plus naturelle de faire de la prise de décision à partir d'un DBN est de passer à une structure de réseau décisionnel dynamique [Zhang, Qi et Poole, 1994] qui a la même structure mais qui explicite les variables de décision et ajoute des utilités à des valeurs de variables ou des transitions. On peut alors extraire une suite de décisions en fonction des observations qui apparaissent.

Présentation

Un réseau décisionnel dynamique (DDN) est à un DBN ce qu'un réseau décisionnel est à un réseau bayésien classique, c'est-à-dire que certaines variables du réseau ne sont pas aléatoires conditionnées par des probabilités, mais contrôlables par une prise de décision extérieure. On peut passer de l'un à l'autre simplement en disant que certaines variables sont des variables de

décision, contrôlables, et que certaines valeurs de certaines variables (ou certaines transitions) apportent récompenses ou coûts.

Formellement, un réseau décisionnel est composé de trois types de nœuds : les variables aléatoires (comme dans un réseau bayésien), les variables de décision, et ces variables peuvent être à valeurs d'utilité. Un réseau décisionnel dynamique a la même structure qu'un DBN à savoir un 2-TBN (voir section III.1). On peut fixer les valeurs des variables de décision, et une séquence de décisions amène à collecter les récompenses ou coûts des variables à valeurs. Ainsi, un critère à optimiser est simplement répercuté sur les valeurs associées aux variables qui sont significatives pour ce critère.

Mise à l'échelle

Trouver une politique optimale peut être fait par des techniques classiques de programmation dynamique [Tatman et Shachter, 1990]. Néanmoins, ce type d'approche ne s'applique pas à de grands problèmes, car on perd l'intérêt d'une représentation factorisée qui n'explicite pas l'espace des états.

D'autre part, il est possible de déplier un DDN sur un horizon donné, de façon à obtenir un arbre solution. Dans un tel arbre, la racine est l'état courant, et chaque décision amène à un nouvel état, et à une nouvelle observation. De cet état, en fonction de la nouvelle observation, une décision doit être prise, etc. Si l'état n'est pas complètement observable, il faut encore une fois utiliser un état courant cru. Pour un tel arbre de petite taille que ce soit en nombre de nœuds ou en profondeur, l'approche la plus classique pour extraire une séquence de décision optimale est la suivante : on peut calculer l'utilité espérée pour un état en faisant remonter les utilités depuis les feuilles de l'arbre, suivant une politique optimiste, pessimiste ou d'espérance probabiliste, par exemple.

On imagine bien que ce type d'approche n'est envisageable que dans quelques cas (non-exclusifs) :

- l'horizon de décision est volontairement limité de façon arbitraire, car le DDN peut être déroulé sur un nombre d'étapes infini (profondeur de l'arbre) ;
- le nombre de couples observations/décision est faible (petit facteur de branchement) car on a une explosion combinatoire si l'on essaie de couvrir tous les couples état/observation possibles à chaque pas de temps ;
- les états ont un structure très particulière qui permet de replier la structure de recherche (recherche de solution factorisée) ; ceci peut être vrai si l'on a des indépendances entre sous-processus, si une observation est influencée par une seule décision, ou qu'une décision conduit de façon certaine à un état donné, etc.

Ces trois conditions sont très peu réalistes, et la solution d'un problème réel doit passer par une résolution approximée.

V.2 Décision en temps contraint et qualité du modèle

Inférence Approchée et Utilité Espérée

La solution utilisée consiste alors en une inférence approchée via un filtre particulaire, mais sans ré-échantillonage des particules car l'on ne dispose pas des observations futures. On projette alors simplement les différentes hypothèses en avant et, pour chaque particule, on collecte les utilités rencontrées.

Dans ce cas, on peut alors ne pas tenir compte des observations futures comme facteur de branchement, et avoir uniquement un arbre basé sur les décisions possibles. Néanmoins, le facteur de branchement d'un tel arbre peut encore être beaucoup trop grand (par exemple dans notre application, on a 144 décisions possibles et plusieurs milliards d'observations différentes possibles ; dérouler toutes les combinaisons est inenvisageable, même sur une profondeur très faible). Et dans le cas général, extraire une séquence de décisions correctes correspond à une recherche de politique optimale, qui est trop complexe pour pouvoir être calculée en ligne.

Une stratégie possible est alors d'évaluer chaque décision sur un horizon donné, dans la logique d'avoir un horizon de décision glissant, et de refaire ce calcul le plus souvent possible. On aura alors une prise de décision de type plutôt réactif, et nous serons capables de trouver une séquence de décisions optimale uniquement sur cet horizon glissant. L'idée est alors de maintenir un état courant cru tout au long du processus (encore une fois sous forme de filtre particulaire) ; et d'évaluer chaque décision séparément pour un horizon de temps donné. On choisira alors la décision qui a la plus grande utilité espérée sur cet horizon, et alors on aura la décision optimale sur cet horizon. Cependant, fixer un horizon "dur" peut être très limitant pour trouver une solution globalement optimale.

Comme on ne sait pas quel est l'horizon de décision qui va donner un comportement satisfaisant, nous proposons d'approximer cet horizon à partir du modèle lui-même. En effet, on peut utiliser une des valeurs des variables comme valeur terminale de la particule. On dira alors qu'une particule atteint un état final si une valeur d'une de ses variables atteint une valeur terminale (dans notre exemple, la variable *comportement général* peut prendre les valeurs terminales **succès** et **échec**, et toute particule dont la variable *comportement général* prend une de ces deux valeur sera dans un état final).

Dans ce cas, il est alors possible de se passer de cette notion d'horizon "dur", et d'inférer les hypothèses pour collecter les utilités jusqu'à ce qu'un état terminal soit atteint. Lorsque toutes les particules ont atteint leur état final, c'est qu'elles ont toutes dépassé l'horizon implicitement contenu dans le modèle. Cette méthode originale permet de gagner en précision dans la prise de décision car l'on se sert d'une information apprise en même temps que le reste du modèle. Il est clair que cette méthode serait beaucoup plus dure à justifier dans le cas d'un modèle donné à priori.

D'autre part, vu que la décision est remise en cause de façon périodique, prendre une décision optimale pour un horizon supérieur à cette période est également non satisfaisant ; il faudrait pouvoir prendre en compte de façon fine ces deux critères distincts. Cependant, nous allons voir que ceci n'est pas la principale difficulté pour la prise de décision.

Est-il possible de tout savoir ?

La prise de décision à partir de modèle statistique appris soulève une question nettement plus fondamentale sur le bien-fondé de se fier entièrement au modèle. En effet, dans notre exemple, le nombre d'observations différentes possibles est de l'ordre de plusieurs milliards, alors qu'une journée entière de collecte de donnée nous permet d'obtenir de l'ordre de 10000 vecteurs d'observations, qui peuvent en plus être redondants. De fait, lorsque l'on veut construire un modèle de grande taille, donc avec un très grand nombre d'observations possibles, on doit souvent le faire avec des données très "creuses", c'est-à-dire que la couverture de l'espace des observations possibles est partielle. Suivant la taille de l'espace, il est possible d'essayer de le couvrir à peu près tout, mais ceci peut également s'avérer impossible dans le cas de corrélations non modélisées (ou non apprises) entre certaines données observables.

De ce fait, les probabilités modélisées dans le DBN sont au mieux dues à quelques observations réelles, et en général sont non-informatives en ce qui concerne des transitions jamais rencontrées (suivant l'algorithme d'apprentissage). On n'est donc pas du tout dans le cas où l'on aurait un bon modèle fiable et correct à partir duquel on veut prendre une décision optimale. De ce fait, l'utilité que l'on va déduire par le mécanisme décrit ci-dessus pour évaluer les décisions est au mieux une légère indication des bonnes décisions. Nous dirons pour parler de ce phénomène que l'on est dans le cas d'un DBN "creux".

Le problème est alors très différent : lorsque l'on infère nos hypothèses pour prédire le futur du système, il est fort probable que les particules "traversent" des transitions de la partie creuse du DBN, et non pas des transitions qui ont été quantifiées lors de l'apprentissage du modèle. Cet état de fait est beaucoup plus préoccupant encore que les problèmes d'horizon de décision.

Facteur de Confiance

Il est alors nécessaire de savoir si l'évaluation d'une décision s'appuie sur une partie creuse du DBN (non apprise) ou sur une partie qui reflète bien un cas qui a fait partie du jeu d'apprentissage. Pour cela, nous proposons d'inclure la mise à jour d'un facteur de confiance lors de l'apprentissage du modèle. Un facteur de confiance doit être attaché à chaque transition du DBN, c'est-à-dire à chaque probabilité conditionnelle d'une variable fille, pour chaque instance de tous ses parents. Pour chaque instance {variable fille, variables parents}, on compte simplement le nombre de mises-à-jour lors de l'apprentissage, c'est-à-dire le nombre de fois que cette instance a été rencontrée dans le jeu d'apprentissage. On a ainsi dans le modèle un indicateur précis de ce qui a été rencontré pendant l'apprentissage par rapport à ce qui ne l'a pas été, ainsi que la fréquence d'apparition de telle ou telle transition dans le jeu de données. Bien utilisé, cet indicateur suffit à faire la différence entre prises de décisions bien informées et peu informées.

Ainsi, pour la prise de décision, pour chaque décision possible, on projette les particules en avant, jusqu'à ce qu'elles atteignent un état final, et on collecte d'une part les utilités espérées, et d'autre part ce facteur de confiance. Pour comparer les décisions, il reste juste à normaliser ces deux valeurs, et choisir suivant le critère qui convient la décision qui semble optimale.

V.3 Décider d'apprendre

Évaluation Multi-Dimensionnelle d'une Décision

Nous avons maintenant une évaluation de chaque décision dans un espace à deux dimensions : l'utilité espérée, et la confiance en cette utilité. Lorsque la confiance est haute, c'est que la décision va très probablement s'appuyer sur un jeu d'apprentissage, alors que lorsqu'elle est faible, la décision va amener à explorer des zones creuses du modèle appris. On peut résumer ceci par la figure V.1.

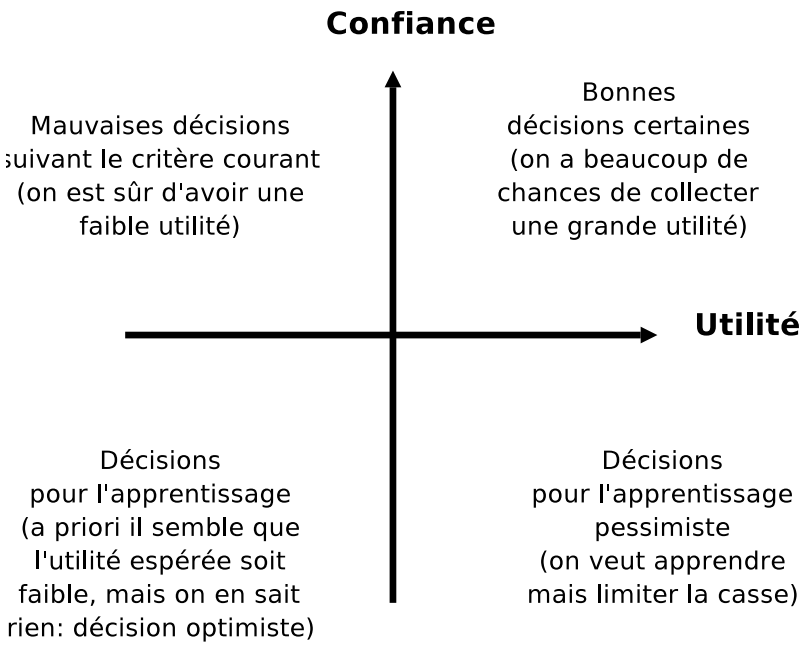


FIG. V.1: Espace de décision

L'intérêt de prendre la décision dans un tel espace est plus grand qu'il n'y paraît de prime abord. En effet, il est clair que nos données d'apprentissage sont généralement trop peu nombreuses pour prendre une décision bien informée. Néanmoins, dans un cadre robotique réaliste, la question du compromis exploration / exploitation est omniprésent. Le robot est en effet soumis à deux critères bien différents : réaliser son objectif de façon optimale, mais aussi compléter ses connaissances actuelles de façon à pouvoir prendre de meilleures décisions dans le futur.

Formaliser ainsi notre espace de décision rend ce compromis particulièrement explicite. Que le robot décide sciemment de prendre un décision à faible facteur de confiance lui permet d'augmenter ses connaissances de façon voulue. On peut alors suivant les cas forcer le robot à augmenter *tout seul* ses connaissances, ou alors privilégier une exécution optimale à court terme.

Au niveau de l'utilité espérée, plusieurs critères peuvent également être pris en compte. Le critère "succès/échec" est bien sûr prépondérant, mais on peut très bien imaginer des sous-critères

permettant de choisir entre plusieurs comportement à succès. Nous y reviendrons dans l'exemple applicatif.

Autres Facteurs : Évaluation de la Connaissance

Si ce facteur de confiance permet d'expliciter le compromis exploration/exploitation de façon originale, d'autres facteurs peuvent être pris en compte pour l'évaluation du modèle courant.

Le premier et le plus simple d'entre eux est de calculer après chaque exécution la probabilité de faire les observations collectées suivant le modèle. Si cette probabilité est trop faible, c'est que le modèle est mauvais, indépendamment de la nature creuse de la base de connaissance. On peut également effectuer des mesures de probabilité à un niveau plus fin, directement en ligne, évaluant la probabilité de la dernière transition effectuée (en fonction de l'état courant cru actuel) par rapport au modèle. Si une transition quasi-impossible suivant le modèle a été faite, c'est que le modèle est encore une fois mauvais.

Ces indicateurs doivent être pris en compte à un niveau supérieur par un méta-contrôleur qui gère le modèle lui-même et qui est responsable de relancer des phases d'apprentissage. L'apprentissage incrémental prendrait ici tout son sens. Ce méta-contrôleur est également en charge de la gestion du compromis exploration/exploitation, suivant les objectifs actuels. Ainsi, ayant une idée de la qualité actuelle du modèle, et des coûts et bénéfices de l'apprentissage, il serait à même de gérer ce compromis.

Idéalement, un tel contrôleur passerait son temps à faire apprendre le robot de la façon la plus rapide possible, et au besoin, pourrait repasser en prise de décisions sécurisée. Il aurait néanmoins à apprendre que certaines phases d'apprentissage trop rapide risqueraient d'être dangereuses pour le robot lui-même (par exemple en cas de collision s'il s'agit d'apprendre des paramètres de navigation).

Ce contrôleur n'est malheureusement pas implémenté en l'état actuel d'avancement. Nous pensons qu'il devrait être appris, ses actions étant le réglage confiance/utilité, ses critères d'utilité seraient les mêmes que pour le contrôleur plus bas niveau, et ses observations celles du contrôleur de bas niveau (en particulier l'utilité effectivement récoltée), mais également des informations diverses sur la qualité du modèle comme celles que nous venons de citer. Il n'aurait pas a priori à tenir compte de variables cachées, et une voie à suivre pourrait être celle de l'apprentissage d'un processus décisionnel de Markov classique.

Approches par apprentissage par renforcement

Ce compromis exploration / exploitation a été largement étudié dans le domaine de l'apprentissage par renforcement pour des problèmes de type processus décisionnels de Markov. Ces approches se proposent d'apprendre une politique optimale pour un problème donné en mettant à jour leur connaissance du problème au fur et à mesure de l'exécution des actions possibles par l'agent. Les méthodes utilisées se découpent en deux grandes familles, les algorithmes sans ou avec modèle.

Approches sans modèles

Les algorithmes de cette première catégorie n'utilisent pas de modèle explicite du contrôleur autre que le problème lui-même, le nombre d'états sur lequel ils travaillent est celui du problème.

Une solution courante consiste à maintenir une valeur Q pour chaque couple (état, action) qui est une approximation de l'utilité espérée de l'exécution de chaque action dans chaque état. La valeur $Q(s, a)$ est mise à jour après chaque exécution de l'action. Cet algorithme fondamental est appelé "Q-learning" [Watkins et Dayan, 1992]. Le choix de l'action a effectuer n'est pas spécifié dans l'algorithme, et de nombreuses solutions ont été proposées. Si l'on ne cherche pas à augmenter la connaissance de l'agent mais uniquement à sélectionner l'action qui donne une utilité espérée maximale, on tombe sur l'algorithme bien connu "value-iteration". La difficulté consiste à choisir une action qui gère le compromis exploration / exploitation.

Une approche extrêmement intéressante se trouve dans [Dearden, Friedman et Russell, 1998]. Les auteurs proposent d'utiliser une valeur Q qui ne soit pas juste une image de l'utilité espérée, mais qui prenne également en compte la valeur espérée de la collecte de nouvelles informations. Cette valeur se base sur des notions de théorie de l'information telle que la valeur de l'information [Howard, 1966]. Dans ce cas, les valeurs Q ne sont plus réelles mais sont des gaussiennes et leur mise à jour nécessite de nombreuses précautions.

Algorithmes avec modèles

Une autre famille d'algorithmes d'apprentissage par renforcement se base sur l'utilisation de modèles très différents du MDP lui-même.

Une extension de l'algorithme "bayesian Q-learning" décrit juste au-dessus propose de d'estimer la valeur Q en se servant en plus d'une distribution sur les différents MDPs sous-jacents du problème [Dearden, Friedman et Andre, 1999]. En effet, si l'on considère toutes les paramétrisations possibles du MDP qui représente le problème, on est capable, au fur et à mesure de l'exploration, de deviner quelles sont les paramétrisations qui représentent la réalité et celles qui en sont loin. La difficulté est alors que le nombre de modèles possibles est infini et qu'il faut utiliser des méthodes d'échantillonnage adaptées basées sur des filtres particulaires, mais même ainsi seuls de petits problèmes peuvent être envisagés.

Une des contributions majeures de ces dernières années est l'algorithme E^3 pour "Explicit Explore or Exploit" [Kearns et Singh, 1998]. Dans cette approche novatrice, l'algorithme se base sur un modèle assez éloigné du MDP à résoudre. Certains états du MDP sont dits "connus", et ceux qui ne le sont pas sont englobés dans un état "inconnu". Lorsqu'un état du MDP à résoudre a été suffisamment exploré, le modèle utilisé par l'algorithme s'enrichit d'un nouvel état connu. L'intérêt de cette approche est de garantir une politique quasiment optimale sur les états connus, et d'offrir une politique permettant de tomber très rapidement dans l'état "inconnu" dans lequel on va pouvoir explorer (en choisissant par des actions au hasard par exemple, ou encore l'action qui a le moins été choisie dans l'état courant). La difficulté est bien entendu de définir le critère qui rend un état connu de façon à pouvoir déduire une politique optimale sur les états connus, et ceci avec une quantité d'explorations polynomiale en la taille du problème. Le critère et les

démonstration de la quasi-optimalité de la politique sur les états connus sont comprises dans l'article.

Cette approche a été étendu à des MDPs sous forme factorisée dans [Kearns et Koller, 1999], de façon à pourvoir traiter des problèmes beaucoup plus grands. Dans ce cas, les auteurs ont démontré que le nombre d'actions nécessaires pour avoir une politique optimale n'est plus polynomial en le nombre d'états mais en le nombre de paramètres du MDP factorisé, qui est généralement exponentiellement plus petit que le MDP décrit de façon classique. Le critère de passage d'un état inconnu à connu est alors sur le nombre de fois qu'une transition a été explorée, ce qui correspond exactement à notre facteur de confiance. Il est a noter que dans ces deux algorithmes, le dilemme exploration / exploitation est rendu explicite par le calcul possible de deux politiques différentes, mais n'est pas résolu.

Une généralisation de E^3 a été proposée sous le nom de R_{max} [Brafman et Tennenholtz, 2001]. Il est défini pour une classe plus large de problèmes, mais surtout propose une méthode simple et efficace pour gérer le dilemme exploration / exploitation. Il propose de calculer une politique optimale (sur les états connus) sur un horizon T, et de recalculer cette politique dès qu'un nouvel état devient connu. Si T est le "mixing time" de la politique , c'est-à-dire le temps nécessaire à la politique pour atteindre son utilité asymptotique, alors l'algorithme donne des garanties d'optimalité. Si ce T est complètement inconnu, on peut exécuter l'algorithme en augmentant T au fur et à mesure. Jusqu'ici, aucun algorithme basé sur un modèle ne propose de façon de guider l'exploration.

Ces approches ne définissent pas comment obtenir une politique optimale sur les états connus (ou les transitions connues). Ceci est fait dans [Guestrin, Patrascu et Schuurmans, 2002], où les auteurs utilisent des techniques classiques de programmation linéaire pour calculer la politique optimale. Cette approche plus pragmatique permet aux auteurs de définir des zones connues ou inconnues de façon plus précise, plus adaptées à l'algorithme de calcul de la politique optimale. Ainsi, au lieu d'avoir des zones connues et inconnues, ils dégagent des zones inconnues (par un calcul de bornes pour la récompense) dans lesquelles l'exploration permettra de façon certaine d'améliorer la politique calculée par l'algorithme de programmation linéaire. On peut résumer grossièrement ceci comme un guidage de l'exploration grâce à l'algorithme de résolution lui-même.

Perspectives

On peut dégager plusieurs réflexions et perspectives de ce qui précède.

Tout d'abord, il faut bien garder en tête que ces approches ne peuvent pas être utilisées telles quelles pour notre problème. En effet, elles sont toutes fortement liées au cadre MDP, où les états sont complètement observables (dans le cadre factorisé : toutes les variables sont observables). Dans ce cas, l'apprentissage est alors simplement une mise à jour des probabilités de transitions. Dans le cas où certaines variables sont cachées, l'apprentissage doit se faire par une mécanique beaucoup plus complexe décrite à la section III.3, et dans le cas général, l'apprentissage est trop lourd pour être effectué en ligne.

Pour notre prise de décision, nous pouvons utiliser le cadre des algorithmes avec modèle, et la distinction transition connue/inconnue peut se faire suivant le critère utilisé dans E^3 (qui est

le même que pour R_{max}). Ceci peut nous permettre de décider quand une transition est suffisamment connue et que sont exploration ne sert plus à rien. Ce critère est important pour obtenir des garanties lors du dilemme exploration/exploitation.

Une idée simple utilisée dans l'algorithme R_{max} pourrait s'appliquer également, c'est l'idée de prendre tout le temps une décision optimale suivant les connaissances courantes, et d'effectuer un nouveau calcul de décision optimale quand une transition devient connue. En effet, nous ne spécifions pas dans notre algorithme quand une décision doit être prise. On pourrait considérer que le moment de prendre une nouvelle décision dépend du nombre de transitions nouvellement connues suivant le critère de E^3.

Par rapport à ces algorithmes, nous avons déjà et de façon simple des décision qui nous amènent à des endroits connus ou inconnus, cette question importante pour ces algorithmes est donc résolue dans notre cadre (par un mécanisme encore une fois très similaire à celui utilisé pour E^3 pour des modèles factorisés).

Un problème important est alors d'explorer aux bons endroits, et deux solutions différentes se dégagent des travaux du domaine. L'approche utilisée par le "bayesian Q-Learning" nous laisse penser que l'évaluation de la *valeur de l'information parfaite*, et sa différence par rapport à l'information actuelle peut permettre d'explorer dans des endroits intéressants, en se basant sur des notions de théorie de l'information. Une approche encore plus intéressante pourrait s'inspirer des travaux de Guestrin et al., il faudrait calculer grâce à notre modèle les endroits qu'il semble intéressant d'explorer, peut-être en utilisant un même système de borne des utilités. Il faut néanmoins garder en tête que notre mécanisme de prise de décision est très différent de celui d'un MDP classique (représenté sous forme de DDN), et qu'obtenir des garanties ou déduire théoriquement les bons endroits où explorer risque d'être difficile.

V.4 Application : optimisation multi-critères d'un comportement robotique

Nous avons implémenté une prise de décision sur les principes que nous avons détaillés. L'architecture générale d'un tel contrôleur est visible sur la figure V.2.

Sur les critères

Nous avons dans cet exemple deux critères à optimiser. L'un porte sur la variable de comportement général, et plus particulièrement sur ses valeurs "succès" et "échec". Nous avons également introduit un critère secondaire beaucoup plus qualitatif sur la variable d'adéquation à l'environnement humain. En effet, certains comportements du robot sont à succès, mais la trajectoire globale ainsi que les vitesses utilisées ne sont pas forcément satisfaisantes. En environnement humain, lorsqu'un robot se déplace, les personnes autour de lui s'attendent à un comportement naturel et intuitif. Si le robot fait de gros détours, ou encore s'il passe très près des personnes, son comportement n'est pas satisfaisant. Les gens autour du robot peuvent se dire que le robot se trompe s'il ne fait pas la trajectoire qui semblerait la plus naturelle, ou encore se sentir agressés si le robot les frôle, et ce même s'il ne les touche pas du tout.

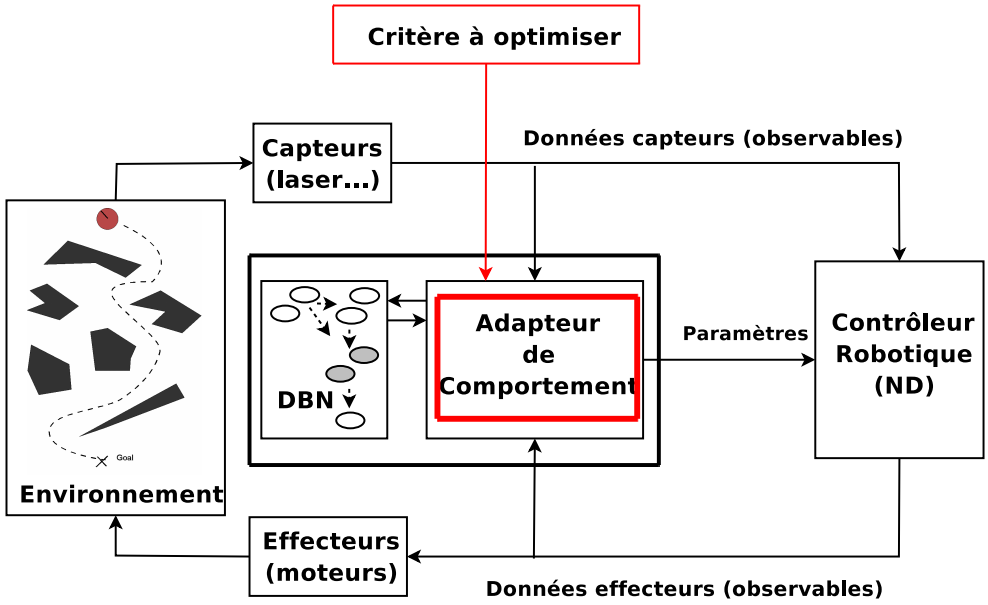


FIG. V.2: Architecture d'un contrôleur robotique

Un autre facteur extrêmement important pour que le robot soit bien accepté porte sur les vitesses et les accélérations utilisées. Si le robot a de grosses accélérations et décélérations, généralement les personnes autour peuvent être surprises (si le robot avance vers une personne pour décélérer très brusquement à un mètre). L'effet inverse est également vrai : si le robot est trop "prudent", il peut devenir ennuyeux, ce qui peut être extrêmement néfaste si le but recherché est l'interaction homme/robot. Un cas simple est que si le robot est dans une zone très encombrée et complexe, on s'attend à ce qu'il aille très lentement (surtout si les obstacles sont humains), mais dès qu'il quitte la zone, il est apprécié qu'il donne une grosse accélération. Si une arrivée en zone ouverte n'est pas suivie par une augmentation très visible de sa vitesse, les personnes interagissant avec lui auront l'impression d'avoir affaire à un robot ne sachant pas naviguer et étant tout le temps très lent et prudent. Dans ce cas de figure, il est très fréquent que des personnes se désintéressent du robot et ne l'attendent pas (cas où le robot est utilisé comme guide de musée).

Ces aspects sont très durs à modéliser du fait de leur nature instinctive et subjective. Pour intégrer de tels aspects dans notre modèle, nous avons demandé à l'opérateur lors des phases d'apprentissage de donner des valeurs qualitatives au comportement du robot parmi "correct", "agressif", "timide".

Les valeurs de cette variable ont ensuite été utilisées comme critère secondaire. Une des difficultés est que l'utilité que nous voulons maximiser est uni-dimensionnelle, et il faut faire attention à l'ordre de grandeur des utilités données. En effet, les valeurs succès et échec sont rencontrées une seule fois par particule, alors que les valeurs de l'adéquation peuvent être rencontrées un grand nombre de fois. Si les valeurs d'utilité données sont comparables, un comportement de forte utilité pourrait être pour le robot d'avoir tout le temps un comportement correct, mais finalement d'échouer dans l'exécution de la tâche. Ceci est à éviter à tout prix. Il faut donc fixer des utilités extrêmement grandes pour le succès de la mission, de façon à n'avoir à choisir qu'entre

des comportements à succès. On pourrait également séparer les critères et les prendre en compte de façon différenciée, c'est à dire en faisant un premier tri de façon a ne garder que les décisions à potentiellement forte probabilité de succès, et trier ceux-ci suivant le critère secondaire. Ces deux approches sont équivalentes si les deux utilités sont incommensurables.

Prise de décision

Pour prendre les décisions, l'algorithme est alors celui de la figure V.3.

```
1.  boucler
2.      si c'est le moment de prendre une décision alors
3.          échantillonner l'état cru courant
4.          pour tous jeux de décisions faire
5.              projeter les particules jusqu'à leur fin sachant le jeu de décisions courant
6.              mémoriser les scores (utilité et confiance)
7.          fin pour
8.          choisir la décision optimale (dans confiance × utilité)
9.      sinon
10.         maintenir l'état courant cru en fonction des observations
11.     fin si
12. fin boucle
```

FIG. V.3: Algorithme général de prise de décision

A la ligne 2, on passe périodiquement dans la branche de la ligne 4. Dans nos expérimentations, la prise de décision a été effectuée toutes les deux secondes. Il est à noter que parfois la prise de décision peut être plus lente que la période de collecte des observations ; dans l'implémentation, deux threads différents s'occupaient de la prise de décision et du maintient de l'état courant cru, qui est effectué à la même fréquence que la collecte des observations ; ces deux tâches s'effectuent donc en parallèle.

La ligne 5, c'est-à-dire l'échantillonnage de l'état courant cru n'est pas forcément indispensable, on peut se contenter de cloner le "belief state" (la représentation de l'état courant cru) pour des raisons de parallélisme. Dans le cas général, néanmoins, pour des contraintes de temps réel, l'estimation de l'état courant cru peut être trop large (en quantité brute de données) pour permettre une prise de décision rapide.

Comme nous l'avons déjà souligné, on peut projeter les particules jusqu'à ce qu'elles atteignent toutes un état final, mais aussi choisir de ne les projeter que sur un horizon donné (ligne 7). Pousser l'inférence de toutes les particules jusqu'à leur fin peut être trop long si l'on n'a pas échantillonné l'état courant cru. Finalement, si la prise de décision doit être faite à très haute fréquence, on peut imaginer échantillonner le filtre particulaire qui sert d'état courant cru et n'inférer les particules que sur un horizon donné. L'échantillonnage lui-même est très rapide.

Il ressort que cet algorithme peut être mis à l'échelle et sur l'horizon et sur la taille des hypothèses, ce qui permet de prendre des décisions quel que soit le temps imparti, et quel que

soit le nombre de décisions à évaluer. Ceci est dû au fait que même si le nombre de décisions est grand, le temps nécessaire pour l'inférence pour chacune d'entre elles est proportionnel au nombre de particules après échantillonnage, et peut être diminué sur un horizon quelconque.

Résultats

Ce contrôleur a été implémenté sur l'instance de l'architecture LAAS [Alami et al., 2000], et le robot rackham présentés à la section III.6. Comme nous l'avons déjà souligné, l'apprentissage du modèle lui-même est trop coûteux pour pouvoir être fait en ligne. En revanche, l'évaluation des 144 différentes décisions possibles est rapide grâce à notre échantillonnage de l'état courant cru. Comme le maintient de l'état courant cru lui-même est tout-à-fait possible en ligne en laissant suffisamment de ressources CPU pour le reste des applications qui marchent sur le robot, la prise de décision réactive en ligne est effective. Pour toutes les décisions, elle prend de l'ordre de la seconde au total sur un pentium®3 850 MHz où marchent en même temps la cartographie de l'environnement (aspect), l'évitement réactif d'obstacle ND, et la collecte des données elle-même.

Lors de la collecte de données, les paramètres contrôlables étaient choisis de façon aléatoire à périodes variables aléatoirement entre 2 et 10 secondes. Ce type de contrôleur très simple amenait très souvent le robot dans des cas d'échecs, que ce soit par collision avec des obstacles, ou alors du fait que le robot reste bloqué (comportement trop prudent dans des environnement fortement encombrés).

Dans une première campagne de test, nous n'avons pas tenu compte du facteur de confiance, et évalué les décisions uniquement suivant leur utilité espérée. Dans ce cas, les échecs sont réduits d'un facteur 10. Les échecs restants apparaissent uniquement dans des cas où la confiance est basse (c'est à dire que le robot a utilisé des paramètres jamais appris dans les environnements correspondants). Dès que l'on utilise le facteur de confiance de façon à ne pas prendre de décision incertaine, les échecs n'apparaissent plus du tout.

Nous avons ensuite essayé d'évaluer le comportement du robot suivant le critère d'adéquation à l'environnement humain. Le problème est que l'on a du mal à évaluer le comportement du robot suivant ce critère hautement subjectif. Si le comportement est effectivement visuellement meilleur, il est dur de le quantifier. Néanmoins, nous avons essayé de préciser un bon comportement, pour voir si le contrôleur le reproduit. En particulier, lorsque le robot sort d'une zone encombrée où il était bon qu'il aille lentement, nous avons étiqueté comme bon les cas où le robot produisait de grosses accélérations au plus tôt en arrivant sur la zone ouverte (cas où le robot interagit avec un humain, une personne ayant tendance à accélérer grandement dans ce cas, et ne devrait pas avoir à attendre le robot). Ce comportement est encore une fois très difficile à quantifier, mais il est nettement reconnaissable dans les expérimentations. Nous concluons alors qu'un tel contrôleur est effectivement capable d'optimiser un comportement robotique suivant les critères qu'on lui donne.

Discussion

On peut conclure de ce qui précède qu'il devient possible de dégager différents comportements pour une même tâche, et de choisir le plus adapté. En effet, on peut mettre une grande utilité pour un bon comportement en environnement humain quand le robot est utilisé dans ce cadre, ou alors ne pas tenir compte de ce facteur et mettre des utilités sur la variation de distance au but fortement négative et sur les vitesses élevées, de façon à minimiser les temps de navigation tout en évitant les cas d'échec.

Lorsque le robot n'est pas utilisé dans l'un ou l'autre cadre, il peut alors essayer au contraire de prendre des décisions l'amenant dans une zone creuse de son modèle, de façon à élargir sa base de connaissances. On peut ici imaginer faire deux types d'apprentissage : l'un supervisé et l'autre non. En effet, un apprentissage supervisé consistera en une collecte des données avec information des données qualitatives subjectives, et un apprentissage non supervisé ou cette donnée sera déduite du modèle courant. La création d'un tel méta-contrôleur qui aurait pour tâche de gérer le modèle, de demander à faire des apprentissages supervisés ou non est une perspective extrêmement importante.

L'évaluation du comportement du robot avec prise de décision n'a pas pu faire l'objet d'études poussées. Comme perspectives, nous pensons utiliser les méthodes décrites dans [Lampe et Chatila, 2006; Held et al., 2006], où les auteurs se sont intéressés de près à ce type de problèmes.

Chapitre VI

Discussions

Nous nous proposons de résumer et conclure cette thèse, ainsi que de soulever des perspectives sur l'apprentissage de modèles stochastiques.

VI.1 Conclusion

Nous avons dans cette thèse fait un rapide tour d'horizon des méthodes de raisonnement dans l'incertain, et avons développé l'apprentissage de modèles stochastiques simples sous forme de modèles de Markov caché. Pour ce type de modèles, la définition des états cachés est un point critique, et nous avons proposé une méthode pour définir ces états en tenant compte à la fois de connaissances imprécises extérieures au système, comme une interprétation humaine, et de connaissances apprises, par classification automatique.

Nous avons ensuite discuté de l'apprentissage de modèles à causalité explicite, plus expressifs que les modèles de Markov caché. Nous avons explicité un algorithme d'apprentissage automatique de réseaux bayésiens dynamiques, là où l'état de l'art se contente de dire "adaptation de l'algorithme classique pour les HMMs". Nous avons fait un tour d'horizon de l'apprentissage de la structure causale, et avons tenté de jeter les bases d'un apprentissage incrémental.

Le chapitre 4 explique comment utiliser ce type de modèle pour la prise de décision en ligne rapide, et ce qu'il faut faire pour tenir compte du fait que le modèle appris est forcément imparfait. Ceci nous a permis d'avancer des idées pour créer un contrôleur chargé de maintenir un modèle cohérent et de gérer des phases d'apprentissage.

Finalement, nous avons montré d'autres utilisations courantes et originales de ce type de modèle, et des perspectives qui nous semblent prometteuses.

VI.2 Perspectives

L'apprentissage de modèles causaux pour des systèmes réels et complexes souffre de nombreuses difficultés. Si les algorithmes eux-mêmes ont été largement étudiés, les autres aspects, qui sont plus spécifiques aux différents domaines on été largement délaissés. Nous avons largement parlé de la définition et de l'apprentissage d'états cachés. Nous avons proposé un algorithme

permettant de garder une sémantique à des états non observables. Nous pensons que cette voie est prometteuse, et que ce type d'approche original devrait pouvoir s'enrichir en utilisant des techniques de classification plus puissantes, types Support Vector Machines. Un autre intérêt de ces méthodes de classification est de rendre implicite la phase d'extraction des données caractéristiques, et donc d'alléger et d'automatiser la phase d'ingénierie indispensable, et généralement discutable.

Une approche complémentaire est celle développée récemment en adaptant la technique de classification dite de l'"information bottleneck" [Elidan et Friedman, 2005]. Cette technique vise à extraire le minimum d'information nécessaire des données d'entrées pour apprendre un réseau bayésien (dynamique ou non) qui explique ces mêmes données. Il nous semble clair que dans cette information minimum peuvent être utilisée des données subjectives d'interprétation, de façon à guider l'apprentissage de structure vers des variables ayant une sémantique claire.

D'autre part, comme nous l'avons largement discuté, suivant les systèmes, un modèle stochastique risque de souffrir de mauvaise qualité si les données d'apprentissage ne sont pas statistiquement suffisantes. Un bon apprentissage incrémental passe par une définition, ou plus exactement une restriction de l'espace des modèles solution. Ici encore, nous pensons qu'une approche contraignant fortement un modèle solution vers des modèles à sémantique claire est indispensable. En effet, le principal problème est que si les variables cachées sont complètement muettes, alors apprendre deux DBNs séparément a peu de chances de conduire à deux modèles donnant le même rôle aux mêmes variables cachées, et alors sommer ces deux modèles est un non-sens. Si l'on est sûr que les deux modèles sont identiques en structure causale alors les quantifications qu'ils contiennent seront uniquement le reflet de nouvelles données d'apprentissage. Si l'on couple cette idée avec celle de la classification visant à extraire le minimum d'information significative des données d'apprentissage, nous pensons que l'on pourra automatiquement créer de nouveaux états reflétant de nouveaux cas appris très différents.

De plus, la technique basée sur l'extraction du minimum d'information se sert de l'incrémentation de la quantité d'information maximale utilisable pour sortir des optimas locaux lors de l'apprentissage quantitatif. Inclure également de nouvelles informations dans ce type de mécanisme pourrait probablement avoir la même fonction.

Dès que l'on disposera d'un bon apprentissage incrémental, développer un système de gestion du modèle suivant les principes que nous avons énoncé sera à notre avis assez rapide. L'utilisation d'un facteur de confiance intégré au modèle permet en effet de prendre des décisions suivant des critères très variés, en utilisant par exemple des techniques classiques de théorie de la décision (dites de "scoring") [Arrow et Raynaud, 1986]. Un tel contrôleur pourrait être construit sur mesure et complètement observable, donnant un processus décisionnel de Markov classique, et serait appris simplement.

Dans des systèmes robotiques, il est très dur de deviner quelles sont les variables significatives pour construire un bon modèle, en particulier en ce qui concerne l'environnement. Un des aspects qui doivent aider est la dynamique relative des différentes variables. Dans notre application à la navigation, les différentes variables observables ont des fréquence de changement

très différentes. Nous pensons qu'une analyse a priori des dynamiques doit permettre de guider dans la sélection des données importantes. Nous proposons également une analyse automatique à travers ce que nous avons appelé réseaux bayésiens à synchronicités multiples, en introduisant des variables cachées servant uniquement d'horloge, de façon à guider l'apprentissage et bien séparer les différentes dynamiques d'un même système. Comme un apprentissage complètement automatique risque d'être très lourd à cause du nombre de variables supplémentaires nécessaires, nous pensons qu'une analyse a priori des dynamiques de façon à ne pas partir d'un réseau complet au niveau des liens causaux peut conduire à de très bons résultats. De plus, si l'algorithme d'apprentissage est suffisamment efficace pour permettre d'avoir un nombre d'horloges capturant toutes les périodicités possibles, alors les modèles obtenus seront à notre avis de très grande qualité.

VI.3 Sur les modèles graphiques, et leur application

Les modèles graphiques, à la base des réseaux bayésiens enrichis ou étendus ont des qualités certaines. La première d'entre elles est qu'ils sont très intuitifs et facilement interprétables. Il faut cependant se méfier de cette interprétation. Comme nous l'avons déjà souligné, un lien causal n'a pas forcément une sémantique de causalité. Dans la définition, ces liens représentent uniquement des probabilités conditionnelles, *qui ne font aucune hypothèse sur les relations causes/conséquences*. La représentation par une flèche peut alors être fort mal adaptée, et conduire à se limiter en terme de formes de modèles possibles. En particulier, on peut refuser intuitivement la présence d'un lien causal entre deux variables, alors que disposer des probabilités conditionnelles et donc jointes entre les deux peut être indispensable. La forme d'un modèle devrait être choisie plus en fonction de ce que l'on voudra calculer avec celui-ci qu'en fonction d'une intuition de causalité.

Cet aspect est également mis en exergue par la présence de variables cachées. Souvent celles-ci sont introduites uniquement pour simplifier des calculs, ou peuvent être introduites car on connaît l'existence d'une causalité, mais on ne peut pas observer la valeur de la variable. Ces deux aspects doivent être considérés comme orthogonaux, et non pas incompatibles. L'intérêt d'un modèle respectant la causalité réelle du système est que celui-ci a de fortes chances d'être strictement minimal en terme de taille du modèle, mais même pas forcément en terme de temps de calculs.

Finalement, si l'inférence pour de tels modèles peut être rapide suivant la forme du modèle, l'apprentissage, quant à lui reste extrêmement lourd et limite fortement la taille des modèles. Comme la plupart des systèmes ne se comportent pas simplement, limiter la taille des modèles et le nombre des interactions entre variables (directes ou par le biais de variables cachées) peut conduire à de très mauvaises approximations. Nous pensons néanmoins que ce type d'approche, couplé à des techniques de classification fines peut permettre de capturer des aspects temporels que peu d'autres modèles permettent de capturer. Contourner l'hypothèse Markovienne (par traitement des données ou par présence de variables d'horloge) est alors indispensable et lourd, mais permet une très large expressivité.

Liste des publications

- **Robel : Synthesizing and Controlling Complex Robust Robot Behaviors**
  par : Benoît Morisset, Guillaume Infantes, Malik Ghallab, Félix Ingrand
  - article dans le 4^e "International Cognitive Robotics Workshop" [CogRob-2004]
  - article court à la conférence ECAI (2004)
- **Robot Introspection through Learned Hidden Markov Models**
  par : Maria Fox, Malik Ghallab, Guillaume Infantes, Derek Long
  article de revue dans Artificial Intelligence, février 2006
- **Apprentissage de modèle d'activité stochastique pour la planification et le contrôle d'exécution**
  par : Guillaume Infantes, Félix Ingrand, Malik Ghallab
  article paru dans la conférence RFIA (2006)
- **Learning Behaviors Models for Robot Execution Control**
  par : Guillaume Infantes, Félix Ingrand, Malik Ghallab
  - article court à la conférence ICAPS (2006)
  - article à la conférence ECAI (2006)
  - article au "Workshop on Planning under Uncertainty and Execution Control for Autonomous Systems" à ICAPS 2006
- **Apprentissage de modèles de comportements pour le contrôle d'exécution en robotique**
  par : Guillaume Infantes, Félix Ingrand, Malik Ghallab
  article dans les journées francophones planification, décision, apprentissage (JFPDA) 2006
- **Rackham : An interactive Robot-Guide**
  par : Aurélie Clodic, Sara Fleury, Rachid Alami, Raja Chatila, Gérard Bailly, Ludovic Brèthes, Maxime Cottret, Patrick Danès, Xavier Dollat, Frédéric Eliseï, Isabelle Ferrané, Matthieu Herrb, Guillaume Infantes, Christian Lemaire, Frédéric Lerasle, Jérôme Manhes, Patrick Marcoul, Paulo Menezes, Vincent Montreuil
  article à la conférence IEEE RO-MAN (2006)

Bibliographie

Alami, R., Chatila, R., Fleury, S., Ghallab, M. et Ingrand, F. [1998]. An Architecture for Autonomy. *International Journal of Robotics Research*, *17*(4), 315-337.

Alami, R., Chatila, R., Fleury, S., Herrb, M., Ingrand, F., Khatib, M. et al. [2000]. Around the lab in 40 days... In *Proceedings of international conference on robotics and automation (ICRA).*

Arrow, K. et Raynaud, H. [1986]. *Social choice and multicriterion decision-making*. Cambridge : MIT Press.

Bengio, S. [2003]. An asynchronous hidden markov model for audio-visual speech recognition. In *Advances in neural information processing systems (NIPS) 15* (pp. 1237–1244).

Bengio, S. [2004]. Multimodal speech processing using asynchronous hidden markov models. *Information Fusion*, *5*(2), 81-89.

Bishop, C. M. [1995]. *Neural networks for pattern recognition*. Oxford University Press.

Bobick, A. et Davis, J. [2001]. The Recognition of Human Movement using Temporal Templates. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, *23*(3), 257-267.

Boyen, X. [2002]. *Inference and learning in complex stochastic processes*. Unpublished doctoral dissertation, Stanford University.

Boyen, X. et Koller, D. [1998]. Tractable Inference for Complex Stochastic Processes. In *Proceedings of the fourteenth Annual Conference on Uncertainty and Artificial Intelligence (UAI).*

Boyen, X. et Koller, D. [1999]. Exploiting the architecture of dynamic systems. In *Proceedings of the sixteenth national conference on artificial intelligence (AAAI).*

Brafman, R. I. et Tennenholtz, M. [2001]. R-MAX — a general polynomial time algorithm for near-optimal reinforcement learning. In *IJCAI* (pp. 953–958).

Brèthes, L. [2005]. *Suivi visuel par filtrage particulaire. application à l'interaction homme-robot*. Unpublished doctoral dissertation, Université Paul Sabatier, Toulouse 3.

Brèthes, L., Lerasle, F. et Danès, P. [2005]. Data fusion for visual tracking dedicated to human-robot interaction. In *Proceedings of International Conference on Robotics and Automation (ICRA)* (pp. 2087–2092).

Bui, H. [2003]. A General Model for Online Probabilistic Plan Recognition. In *Proceedings of international joint conference on artificial intelligence (IJCAI).*

Bui, H., Venkatesh, S. et West, G. [2002]. Policy Recognition in the Abstract hidden Markov model. *Journal of AI Research*, *17*, 451-499.

Chickering, D. [1996]. Learning bayesian networks is NP-complete. *Learning from Data :*

Artificial Intelligence and Statistics, *V*, 121-130.

Chickering, D. [2002]. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, *3*, 507-554.

Clodic, A., Fleury, S., Alami, R., Herrb, M. et Chatila, R. [2005]. Supervision and interaction. In *Proceedings of International Conference on Advanced Robotics (ICAR)* (p. 725-732).

Cohen, I., Sebe, N., Chen, L., Garg, A. et Huang, T. S. [2003]. Facial Expression Recognition from Video Seqences : Temporal and Static Modelling. *Computer Vision and Image Understanding : Special Issue on Face Recognition*, *91*, 160-187.

Cooper, G. et Herskovits, E. [1992]. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*(4), 309-347.

Cristianini, N. et Shawe-Taylor, J. [2000]. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press.

Cuadras, C. M., Fortiana, J. et Oliva, F. [1997]. The proximity of an individual to a population with applications in discriminant analysis. *Journal of Classification*, *14*(1), 117-136.

Dean, T. et Kanazawa, K. [1990]. A model for reasoning about persistence and causation. *Computational Intelligence*, *5*(3), 142–150.

Dearden, R., Friedman, N. et Andre, D. [1999]. Model based bayesian exploration. In *Proceedings of fifteenth conference on uncertainty in artificial intelligence (UAI)* (pp. 150–159).

Dearden, R., Friedman, N. et Russell, S. [1998]. Bayesian Q-Learning. In *Proceedings of the national conference on ai (aaai-98).*

Doucet, A., Freitas, N. de, Murphy, K. et Russell, S. [2002]. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of Annual Conference on Uncertainty and Artificial Intelligence (UAI)* (pp. 176–183).

Elidan, G. et Friedman, N. [2001]. Learning the dimensionality of hidden variables. In *Proceedings of the seventeenth Conference on Uncertainty in Artificial Intelligence (UAI).*

Elidan, G. et Friedman, N. [2005]. The information bottleneck approach. *Journal of Machine Learning Research*, *6*, 81-127.

Elidan, G., Lotner, N., Friedman, N. et Koller, D. [2000]. Discovering hidden variables : A structure-based approach. In *Advances in neural information processing systems (NIPS 2000).*

Elidan, G., Ninio, M., Friedman, N. et Schuurmans, D. [2002]. Data perturbation for escaping local maxima in learning. In *Proceedings of eighteenth national conference on artificial intelligence (AAAI).*

Fleury, S., Herrb, M. et Chatila, R. [1997]. GenoM : A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture. In *Proceedings of international conference on Intelligent Robots and Systems (IROS),* (Vol. 2, p. 842-848).

Fox, M., Ghallab, M., Infantes, G. et Long, D. [2006, février]. Robot introspection through learned hidden markov models. *Artificial Intelligence*, *170*(2), 59-113.

Friedman, N. [1998]. The bayesian structural EM algorithm. In *Proceedings of Uncertainty in Artificial Intelligence (UAI).*

Friedman, N. et Koller, D. [2003]. Being Bayesian about Bayesian network structure : A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, *50*(1–2), 95–125. (Full version of UAI 2000 paper)

Friedman, N., Murphy, K. et Russell, S. [1998]. Learning the structure of dynamic probabilistic networks. In *Proceedings of the fourteenth conference on uncertainty in artificial intelligence (uai).*

Friedman, N., Nachman, K. I. et Peér, D. [1999]. Learning bayesian networks structure from massive datasets : The " sparse candidate" algorithm. In *Proceedings of the fifteenth conference on uncertainty in artificial intelligence (UAI).*

Gonzales, C. et Jouve, N. [2006]. Apprentissage de structure de reseaux bayesiens a partir de reseaux markoviens. In *15e congrès francophone Reconnaissance des Formes et Intelligence Artificielle (RFIA).*

Gordon, N. J. [1994]. *Bayesian methods for tracking*. Unpublished doctoral dissertation, Imperial College, University of London.

Gordon, N. J., Salmond, D. J. et Smith, A. F. M. [1993]. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, *140*(2), 107-113.

Guestrin, C., Patrascu, R. et Schuurmans, D. [2002]. Algorithm-Directed Exploration for Model-Based Reinforcement Learning in Factored MDPs. In *Proceedings of International Conference on Machine Learning (icml)* (p. 235-242).

Hauser, A., Sottas, P.-E. et Millán, J. [2002]. Temporal processing of brain activity for the recognition of EEG patterns. In *Proceedings of International Conference on Artificial Neural Networks.*

Held, J., Lampe, A. et Chatila, R. [2006]. Linking mobile robot performances with the environment using system maps. In *Proceedings of International Conference on Intelligent Robots and Systems (IROS).*

Howard, R. A. [1966]. Information value theory. *IEEE Transaction on Systems Science and Cybernetics*, *2*, 22-26.

Howard, R. A. et Mathson, J. E. [1984]. Readings on the principles and applications of decision analysis. In R. A. Howard et J. E. Mathson (Eds.), (pp. 721–762). Strategic Decisions Group, Menlo Park, California.

Isard, M. et Blake, A. [1996]. Contour tracking by stochastic propagation of conditional density. In *Proceedings of fourth european conference on computer vision.*

Kanazawa, K., Koller, D. et Russel, S. J. [1995]. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proceedings of the eleventh conference on uncertainty and artificial intelligence (UAI).*

Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R. et Wu, A. [2002, Juillet]. An efficient k-means clustering algorithm : analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(7), 881-892.

Kearns, M. et Koller, D. [1999]. Efficient Reinforcement Learning in Factored MDPs. In *Proceedings of international joint conference on ai (IJCAI).*

Kearns, M. et Singh, S. [1998]. Near-optimal reinforcement learning in polynomial time. In *Proceedings of international conference on machine learning (ICML).*

Kohonen, T. [1984]. *Self-organisation and associative memory*. Springer-Verlag.

Koller, D. et Fratkina, R. [1998]. Using learning for approximation in stochastic processes. In *Proceedings of International Conference on Machine Learning (ICML).*

Lampe, A. et Chatila, R. [2006]. Performance measure for the evaluation of mobile robot autonomy. In *Proceedings of International Conference on Robotics and Automation (ICRA).*

Levine, G. et DeJong, G. [2006]. Explanation-based acquisition of planning operators. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS).*

Liao, L., Fox, D. et Kautz, H. [2004]. Learning and Inferring Transportation Routines. In *Proceedings of national conference on artificial intelligence (AAAI).*

Liao, L., Fox, D. et Kautz, H. [2005]. Location-based activity recognition using relational markov networks. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI.*

Liu, J. et Chen, R. [1998]. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association*, *93*, 1022-1031.

MacEachern, S. N., Clide, M. et Liu, J. S. [1999]. Sequential importance sampling for nonparametric bayes models : the next generation. *Canadian Journal of Statistics*, *27*, 251-267.

Millán, J., Renkens, F., Mouriño, J. et Gerstner, W. [2004]. Brain-actuated interaction. *Artificial Intelligence*.

Minguez, J., Osuna, J. et Montano, L. [2004]. A "Divide and Conquer" Strategy based on Situations to achieve Reactive Collision Avoidance in Troublesome Scenarios. In *Proceedings of International Conference on Robotics and Automation (ICRA).*

Morisset, B. [2002]. *Vers un robot au comportement robuste. apprendre à combiner des modalités sensori-motrices complémentaires.* Unpublished doctoral dissertation, Université Paul Sabatier, Toulouse.

Munteanu, P. et M.BEndou. [2001]. The EQ framework for learning equivalence classes of bayesian networks. In *Proceedings of the IEEE international conference on data mining* (p. 417-424).

Nam, Y. et Wohn, K. [1996]. Recognition of Space-Time Hand Gestures using Hidden Markov Models. In *Acm symposium on virtual reality software and technology* (pp. 51–58).

Oates, T., Schmill, M. D. et Cohen, P. R. [2000]. A method for clustering the experiences of a mobile robot that accords with human judgements. In *Proceedings of the 17th national conference on artificial intelligence (AAAI)* (pp. 846–851).

Pearl, J. [2000]. *Causality : models reasoning and inference.* (C. U. Press, Ed.).

Rabiner, L. R. [1989, février]. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, *77*(2), 257-286.

Rebsamen, B., Burdet, E., Guan, C., Zhang, H., Teo, C. L., Zeng, Q. et al. [2006]. A Brain-Controlled Wheelchair Based on P300 and Path Guidance. In *IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob).*

Rubin, D. [1988]. Using the sir algorithm to simulate posterior distributions. *Bayesian Statistics*, *3*, 395-402.

Russell, S. et Norvig, P. [2003a]. *Artificial intelligence : a modern approach* (P. Hall, Ed.).

Russell, S. et Norvig, P. [2003b]. Artificial intelligence : a modern approach. In P. Hall (Ed.), (pp. 625–628).

Smyth, P., Heckerman, D. et Jordan, M. [1997]. Probabilistic independence networks for hidden markov probability models. *Neural Computation*, *9*, 227-269.

Spirtes, P., Glymour, C. et Scheines, R. [1993]. *Causation, prediction and search*. Springer Verlag.

Tatman, J. et Shachter, R. [1990]. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, *20*(2), 365-379.

Teichteil-Königsbuch, F. [2005]. *Approche symbolique et heuristique de la planification en environnement incertain : optimisation d'une stratégie de déplacement et de prise d'information*. Unpublished doctoral dissertation, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace (ENSAE : Sup'Aéro).

Vapnik, V. [1998]. *Statistical learning theory*. John Wiley.

Watkins, C. J. et Dayan, P. [1992]. Q-learning. *Machine Learning*, *8*(3), 279-292.

Wilson, A. et Bobick, A. [1999]. Parametric Hidden Markov Models for Gesture Recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, *21*(9), 884–900.

Wilson, A. et Bobick, A. [2001]. Hidden Markov Models for Modeling and Recognizing Gesture Under Variation. *International Journal of Pattern Recognition and Artificial Intelligence*, *15*(1), 123–160.

Yoon, H., Soh, J., Bae, Y. J. et Yang, H. S. [2001]. Hand gesture recognition using combined features of location, angle and velocity. *Pattern Recognition*, *34*, 1491-1501.

Zhang, N., Qi, R. et Poole, D. [1994]. A computational theory of decision networks. *International Journal of Approximate Reasoning*, *11*(2), 83-158.

Table des figures

Liste des tableaux