N° d'ordre : 439 Année 2005

THÈSE

présentée en vue de
l'obtention du titre de

DOCTEUR
de
L'ÉCOLE NATIONALE SUPÉRIEURE
DE L'AÉRONAUTIQUE ET DE L'ESPACE

ÉCOLE DOCTORALE : SYSTÈMES

SPÉCIALITÉ : Intelligence artificielle

Florent TEICHTEIL-KOENIGSBUCH

Approche symbolique et heuristique de la planification en environnement incertain : optimisation d'une stratégie de déplacement et de prise d'information

Soutenue le 8 décembre 2005 devant le jury :

MM.	F.	CHARPILLET	Président
	R.	CHATILA	
	P.	FABIANI	Directeur de thèse
	A.I.	MOUADDIB	Rapporteur
	R.	SABBADIN	
	O.	SIGAUD	Rapporteur

Thèse préparée au sein du département commande des systèmes et dynamique du vol (DCSD) de l'ONERA-centre de Toulouse.

THÈSE

présentée en vue de
l'obtention du titre de

DOCTEUR
de
L'ÉCOLE NATIONALE SUPÉRIEURE DE L'AÉRONAUTIQUE ET DE L'ESPACE

SPÉCIALITÉ : INTELLIGENCE ARTIFICIELLE

par

Florent TEICHTEIL-KÖNIGSBUCH

Approche Symbolique et Heuristique de la Planification en Environnement Incertain : Optimisation d'une Stratégie de Déplacement et de Prise d'Information

Soutenue le 8 décembre 2005 devant la Commission d'Examen :

Rapporteurs	**M. Abdel-Illah**	**Mouaddib**
	M. Olivier	**Sigaud**
Examinateurs	**M. François**	**Charpillet**
	M. Régis	**Sabbadin**
	M. Raja	**Chatila**
	M. Patrick	**Fabiani**

À ma famille,
À Raoudha,
À mes amies et amis,
À Patrick

Table des matières

Introduction

> *« L'Intelligence Artificielle est un domaine des sciences et de l'ingénierie qui s'intéresse à la compréhension informatique de ce qui est communément appelé* comportement intelligent*, et à la création d'artifices qui exhibent de tels comportements. »*

Cette définition de l'Encyclopédie de l'Intelligence Artificielle [1] souligne l'ambiguïté du terme "*intelligence*" dans l'expression quelque peu fantasmagorique "*intelligence artificielle*". En effet, elle remet précautionneusement en cause le vieux fantasme prométhéen qui consisterait à pourvoir des machines artificielles d'intelligence comparable, voire supérieure, à celle de l'Homme.

L'informatique ne serait donc qu'un moyen de comprendre les comportements intelligents, et de les reproduire. En aucun cas, elle permettrait de créer *par elle-même* de tels comportements. Nous sommes donc actuellement bien loin des préoccupations des fondateurs de l'Intelligence Artificielle, tel Marvin Lee Minsky, selon lequel l'Intelligence Artificielle consisterait à produire des programmes visant à dépasser la pensée humaine [2]. Ce dernier a d'ailleurs changé d'avis : il défend désormais l'idée que la logique informatique ne s'applique pas au monde réel [3].

Ainsi, de nos jours, l'objectif de l'intelligence artificielle est plus modeste. Le terme *intelligence* ne doit pas être pris au sens propre, dans la mesure où il ne s'agit pas de créer des programmes imaginatifs dotés d'une intelligence créatrice, qui leur permettrait de prendre des initiatives nouvelles en présence de situations inconnues. L'intelligence des programmes informatiques doit être plutôt considérée comme la capacité à prendre des décisions raisonnées et rationnelles, dans un contexte et suivant un principe de raisonnement connus. Aussi, un programme sera en mesure de s'adapter à une situation inhabituelle uniquement si son concepteur a prévu cette situation éventuelle, ou s'il lui a donné les moyens de l'analyser et de la comprendre.

Dans ce cadre restrictif mais déjà très ambitieux, la planification d'actions est un domaine de l'intelligence artificielle qui vise à calculer automatiquement les meilleures actions que doit effectuer un agent, afin de réaliser une tâche ou un but donnés, en maximisant éventuellement un critère d'utilité. L'agent ne trouve pas de lui-même la démarche intellectuelle qui lui permet de réaliser cette tâche, mais l'être humain lui indique la manière de réfléchir pour y parvenir, sous la forme d'un algorithme informatique. Ainsi, l'agent dispose d'un algorithme — une façon de penser — grâce auquel il peut résoudre le problème de planification plus vite et plus facilement que l'être humain.

Il existe des algorithmes qui donnent les moyens à l'agent d'optimiser sa stratégie d'actions en même temps qu'il observe et qu'il apprend l'environnement. Cependant, il ne peut déchiffrer et comprendre ses observations de l'environnement que s'il connaît les différentes structures et dynamiques possibles de celui-ci. Ainsi, la planification d'actions par apprentissage doit être considérée comme une adaptation dynamique des algorithmes de planification statique, et non comme un surcroît d'intelligence pour l'agent.

Aussi, dans cette thèse, nous nous limiterons au calcul optimal de stratégies de déplacement et de prise d'information dans un environnement dont la dynamique et la structure sont connus à l'avance. Ainsi, il ne s'agit pas d'améliorer la stratégie courante de l'agent au fur et à mesure que l'environnement est appris, en supposant qu'il correspond parfaitement à un modèle typique d'environnement que l'agent est en mesure d'analyser. Nous nous intéresserons toutefois à des algorithmes qui prennent en compte les incertitudes de l'environnement, au point que la stratégie d'actions est multiple suivant les situations stochastiques éventuelles dans lesquelles l'agent peut se retrouver. Bien entendu, la stratégie d'actions sera incapable de réagir à un changement imprévu de la dynamique de l'environnement.

Si ce cadre d'étude peut paraître bien modeste, nous verrons que le calcul de stratégies d'actions optimales dans ce contexte limité est déjà très difficile, en raison de l'explosion combinatoire dans le choix des stratégies d'actions possibles. Nous appuierons notre discours sur des missions de déplacement et de prise d'information issues du projet ReSSAC de l'ONERA [4, 5]. Ce projet concerne un hélicoptère autonome qui évolue dans un environnement incertain, dans le but de rechercher et de prêter assistance à des personnes en danger. L'espace géographique est partitionné en régions ou zones dans lesquelles se trouvent éventuellement les personnes recherchées. Certaines zones sont dangereuses si bien que l'hélicoptère doit les explorer avec précaution : autrement dit, le problème de planification d'actions consiste à trouver un compromis optimal entre, d'une part, le risque encouru, et d'autre part le nombre de personnes à secourir, ainsi que l'ordre de sauvetage.

La figure 1 représente la relation entre le *planificateur*, qui optimise la stratégie, et la *commande*, qui effectue au mieux les actions de la stratégie en fonction d'une estimation de l'état courant. La stratégie est optimisée en *boucle ouverte*, c'est-à-dire sans interaction avec l'environnement, mais elle prend en compte les changements stochastiques possibles de ce dernier. Elle est ensuite jouée par le *superviseur* en boucle fermée : le superviseur interroge la stratégie, afin de connaître l'action optimale à effectuer. Il envoie l'ordre à la commande de réaliser cette action à l'aide de fonctions de guidage et de pilotage, telles "rejoindre des objectifs de navigation" ou "prendre des images coordonnées avec les déplacements". La durée Δt d'une action est variable, mais les changements d'état sont synchronisés par le superviseur.

Dans une première partie, nous présenterons des algorithmes existants pour résoudre ce type de problèmes. Bien que les techniques modernes d'optimisation combinatoire sont de plus en plus efficaces, elles se heurtent encore à des difficultés de modélisation et de résolution pour des problèmes de très grande taille. Ainsi, dans une deuxième partie, nous proposerons un algorithme qui réduit automatiquement la taille du problème à résoudre, afin de faciliter son optimisation. Néanmoins, cette approche ne simplifie pas la modélisation du problème, qui est souvent laborieuse voire impossible dans certains cas. Par conséquent, dans une troisième partie, nous

proposerons un modèle hiérarchique et générique de problèmes de planification stochastique, qui permet de modéliser très simplement des problèmes de grandes tailles.

Enfin, il arrive souvent que les situations initiales possibles de l'agent soient connues avant l'optimisation de sa stratégie d'actions. Ainsi, dans le cas de l'hélicoptère autonome du projet ReSSAC, la zone de décollage, l'autonomie de l'hélicoptère et les conditions météorologiques sont des exemples de situations initiales connues. Sous cette hypothèse, l'optimisation de la stratégie d'actions ne nécessite pas d'explorer toutes les situations possibles. Aussi, dans une quatrième partie, nous présenterons une dernière contribution, consistant en une classe d'algorithmes symboliques qui optimisent une stratégie partielle sur un sous-ensemble atteignable de situations possibles, et focalisée sur les tâches à accomplir.

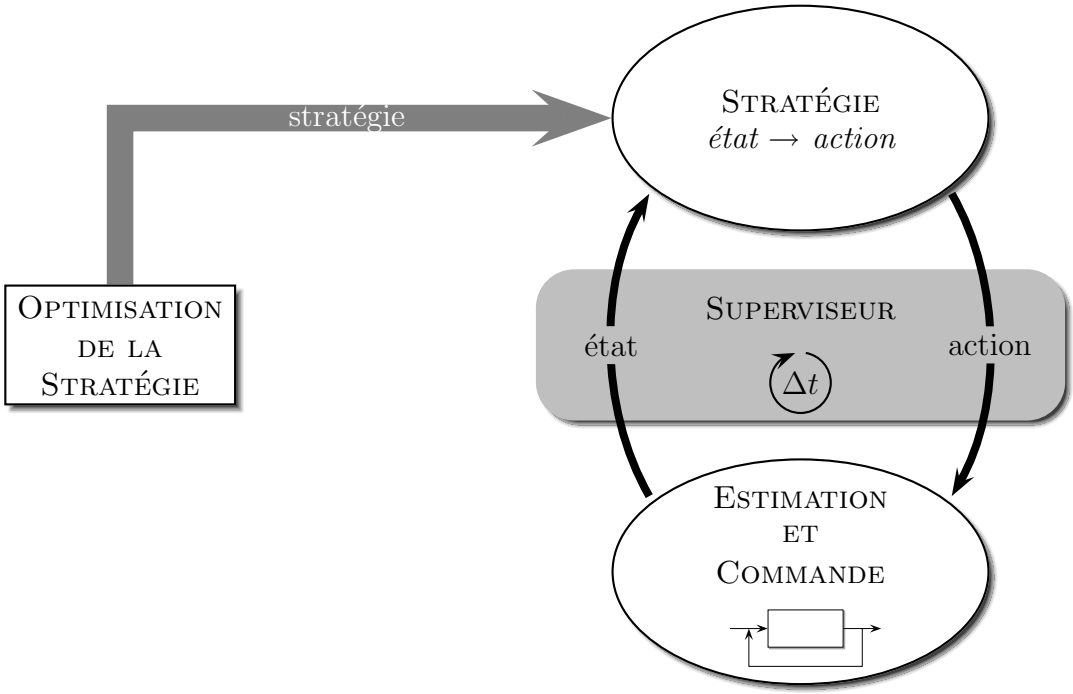


Fig. 1 – Supervision en boucle fermée d'une stratégie optimisée en boucle ouverte

Première partie

Planification d'actions en environnement incertain

Motivation

La planification est un domaine de l'intelligence artificielle qui vise à produire des *plans* pour un agent autonome, c'est-à-dire un ensemble d'actions qu'il doit réaliser afin de rejoindre un état but donné en partant d'un état initial connu. La planification se divise elle-même en deux sous-domaines différents :

- planification *déterministe* ou *classique* : les effets des actions (état successeur) sont supposés déterministes ;
- planification *stochastique* : les effets des actions sont stochastiques, c'est-à-dire que les états successeurs d'un état donné pour une action donnée sont une distribution de probabilité.

Planification déterministe

En planification déterministe, un plan définit un seul chemin partant de l'état initial et arrivant à l'état but. De nombreux algorithmes ont été développés dans le but de produire le plus efficacement possible un tel plan [6, 7]. Un critère d'optimisation peut éventuellement être pris en compte, tel la maximisation des gains cumulés sur le chemin menant de l'état initial à l'état but. La figure 2 représente un plan solution produit avec des actions déterministes, dans une représentation de graphe où les nœuds sont les états de l'agent et les arêtes sont ses actions.

L'espace de recherche des plans solutions est soit l'espace d'états de l'agent, soit l'espace des plans. La recherche dans l'espace d'états peut être effectuée en arrière depuis l'état but (exemple de l'algorithme de `Dijkstra` [8]), ou en avant depuis l'état initial (exemple de l'algorithme `A`* [9]). Depuis une dizaine d'années, la recherche en avant dans l'espace des plans s'est imposée en planification déterministe, grâce à l'algorithme `GraphPlan` [10], qui permet de trouver rapidement un plan solution pour des problèmes complexes, définis sous formes de propositions logiques et de prédicats. `GraphPlan` développe en avant un graphe des états atteignables puis, une fois le but atteint, il recherche en arrière un chemin faisable entre l'état initial et l'état but. D'autres approches utilisent une représentation du problème sous de satisfaction de contraintes propositionnelles (exemple de `SatPlan` [11]). Afin de faciliter la recherche

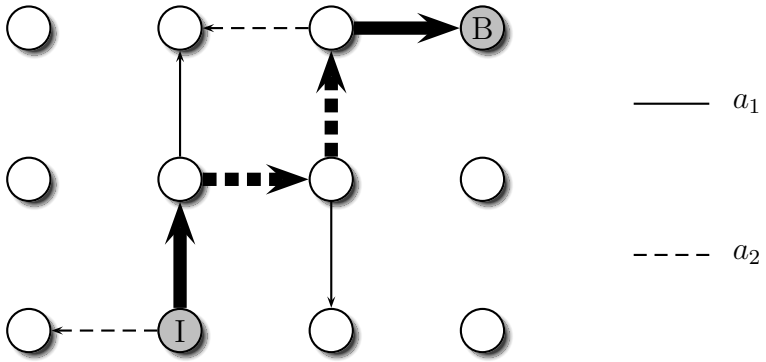


FIG. 2 – Exemple de plan produit avec des actions déterministes : I est l'état initial, et B est l'état but

du plan solution, ces algorithmes utilisent pour la plupart une *heuristique*, qui guide le développement du chemin vers l'état but à l'aide de critères empiriques.

Planification stochastique

Si les actions ont des effets stochastiques, un état peut avoir plusieurs successeurs possibles, pondérés par une probabilité de les atteindre pour chaque action. Ainsi, contrairement à la planification déterministe, il n'est pas raisonnable de construire un plan statique d'actions, c'est-à-dire un seul chemin menant de l'état initial à l'état but, puisque l'agent risque de s'écarter du chemin à cause des effets stochastiques des actions (cf. figure 3). Deux approches différentes sont généralement adoptées pour résoudre ce problème :

- planification *réactive* : un plan d'actions est produit et appliqué temporairement tant que les actions n'échouent pas ou que l'environnement ne change pas ; une procédure de replanification est lancée dès que l'effet d'une action du plan est différent de celui prévu dans le plan (le plan échoue).
- planification *proactive* : un ensemble de plans statiques est produit entre tous les états de l'agent et l'état but : de manière duale, une fonction appelée *politique* ou *stratégie* est calculée, qui associe une action à réaliser dans chaque état de l'agent.

Les approches de planification *réactive* utilisent souvent des techniques de planification déterministe pour générer localement un plan temporaire [12, 13]. Cependant, si un critère d'optimisation doit être considéré, ces approches n'intègrent pas les probabilités d'échec des actions dans le calcul optimal de la stratégie. Ce choix repose sur la difficulté de modéliser les probabilités d'échec des actions, inconnues pour certains problèmes, et sur la difficulté de calculer des stratégies qui prennent

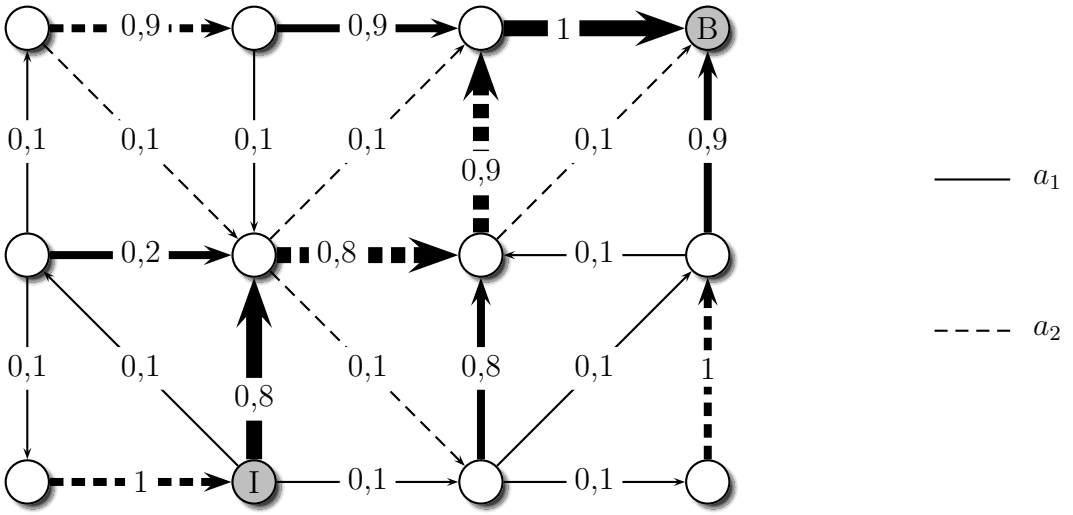


FIG. 3 – Exemple de politique obtenue avec des actions stochastiques : I est l'état initial, et B est l'état but

en compte les probabilités d'échec dans l'optimisation du critère. En effet, le calcul de ces stratégies nécessite d'explorer tout l'espace d'états.

Néanmoins, le calcul de stratégies basées sur les probabilités d'échec des actions a le mérite de produire des politiques optimales, puisque la valeur du critère optimisé est pondérée par les probabilités d'échec des actions. Aussi, ces approches sont souvent utilisées pour la planification en environnement incertain. Les Processus Décisionnels de Markov (MDP) [14] sont un modèle de planification proactive basé sur la programmation dynamique stochastique [15], qui prend en compte les effets probabilistes *markoviens* des actions dans l'optimisation du critère.

Si l'avantage des MDP réside dans la robustesse des politiques produites aux aléas de l'environnement, ils ont le défaut d'explorer tout l'espace d'états, ce qui peut être rédhibitoire pour des problèmes de grande taille [16, 17, 18, 19, 20, 21]. Malheureusement, ainsi que nous le verrons plus loin, l'espace d'états des problèmes de déplacement et de prise d'information, tels les missions RESSAC, est souvent de très grande taille. Toutefois, des méthodes hiérarchiques et structurées, basées sur certaines caractéristiques de l'espace d'états, ont été récemment proposées afin de faciliter l'optimisation de la stratégie sur l'ensemble des états. Aussi, dans cette partie, nous présenterons la décomposition [22, 23, 24, 25] et la factorisation [26, 27, 28, 29, 30, 31, 32, 33] de MDP, dont nous discuterons des avantages pour les problèmes de déplacement et de prise d'informations. Ces deux techniques seront ensuite utilisées dans toutes les parties suivantes.

Nous supposerons que la stratégie est calculée entièrement avant le début de la mission. Si le temps est une ressource limitée au point que la stratégie ne peut pas être optimisée entièrement, il existe des méthodes dites *anytime* qui calculent une stratégie sous-optimale sur un créneau de temps disponible mais limité. La stratégie est améliorée incrémentalement dès qu'un créneau de calcul se libère [34].

Précisons enfin que les MDP ne prennent pas en compte l'incertitude sur les états observés. Une extension des MDP, appelée POMDP (MDP Partiellement Observables), permet de construire des stratégies robustes aux incertitudes d'observation [35, 36, 37]. Les POMDP sont beaucoup plus difficiles à résoudre que les MDP car l'espace d'états des croyances, intégré dans le calcul de la stratégie, est continu. De plus, nous nous limitons dans cette thèse à l'étude de problèmes de déplacement et de prise d'information où l'observation des états de navigation est totale à l'échelle du planificateur, et où la position des informations dans l'environnement est connue au moment de l'optimisation de la stratégie. Dans le cas contraire, et si aucune distribution de probabilité sur les états observés est connue a priori, des techniques d'apprentissage par renforcement [38] permettent d'optimiser la stratégie au fur et à mesure de l'apprentissage du modèle. D'autres approches, dont certaines sont basées sur des modèles de Markov cachés, visent à apprendre l'environnement géographique en cours de mission [39, 40].

Notons également que le cadre des MDP se restreint à des lois de probabilité markoviennes. Des techniques ont donc été développées dans le but d'optimiser des stratégies dont les effets des actions ne sont pas markoviens [41]. Plus généralement, il existe une extension possibiliste des MDP, où les incertitudes et les préférences de l'agent sont représentées par des lois possibilistes [42]. Enfin, l'espace d'états étant supposé discret, des approches ont été proposées afin de résoudre des MDP à états continus [43, 44].

Plan de la partie I

Chapitre 1 : formalisme des Processus Décisionnels de Markov et algorithmes d'optimisation

Chapitre 2 : présentation de deux modèles structurés de MDP :
- *décomposition* de MDP énuméré (optimisation de sous-MDP de petites tailles, définis dans des partitions de l'ensemble d'états total)
- *factorisation* de MDP (espace d'états défini implicitement ou en intention par des caractéristiques)

Chapitre 1

Processus Décisionnels de Markov

1.1 Chaînes de Markov

1.1.1 Un modèle de processus dynamique stochastique sans mémoire du passé

Les Processus Décisionnels de Markov s'appuient sur la théorie mathématique des chaînes de Markov, qui permettent de décrire l'évolution d'un processus stochastique sans mémoire du passé. Comme l'indique la définition 1, une chaîne de Markov est une suite discrète d'états stochastiques, tels que la probabilité d'obtenir un état à un instant donné ne dépend que de l'état précédent de la chaîne. Ainsi, la dynamique du système n'intègre pas la mémoire de l'historique des états passés.

Définition 1 Chaîne de Markov

Soit $\mathcal{E}$ l'ensemble des états et T l'ensemble des instants d'un processus dynamique.
Une chaîne de Markov est une suite $(E_t)_{t \in T}$ de variables aléatoires de $\mathcal{E}$ telle que, pour tout instant t dans T et tout historique d'états $(e_0, \ldots, e_t) \in \mathcal{E}^T$:

$$\forall\, e_{t+1} \in \mathcal{E}\ ,\ P\left(E_{t+1} = e_{t+1} \mid E_0 = e_0, \ldots, E_t = e_t\right) = P\left(E_{t+1} = e_{t+1} \mid E_t = e_t\right)$$

La probabilité précédente vérifie la *propriété de Markov*. Elle définit une fonction de $\mathcal{E} \times T$ dans $\mathcal{E}$, appelée *fonction de transition* $\mathcal{T}$ de la chaîne de Markov.

Lorsque l'ensemble d'états est de dimension finie, il est possible de représenter la chaîne de Markov par un graphe dont les nœuds sont les états du processus et les arêtes sont pondérées par les probabilités de transition entre les états. Si, de plus, la chaîne de Markov est stationnaire, c'est-à-dire que les probabilités de transition ne dépendent pas du temps, la fonction de transition ne dépend pas de T : c'est une

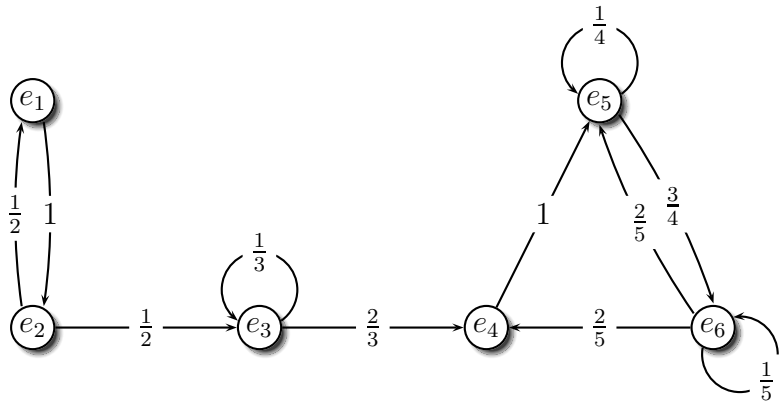


FIG. 1.1 – Exemple de chaîne de Markov de dimension finie

fonction de $\mathcal{E}$ dans lui même qui peut se représenter sous forme d'une matrice carrée, dont les lignes sont les états courants et les colonnes sont les états successeurs.

Sous forme matricielle, la dynamique de la chaîne de Markov s'écrit : $E' = \mathcal{T} \cdot E$, où E est l'ensemble des états courants possibles, et E' est l'ensemble des états successeurs. Par exemple, la matrice de transition de la chaîne de Markov définie par le graphe de la figure 1.1 est :

$$\mathcal{T} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{2}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 0 & \frac{2}{5} & \frac{2}{5} & \frac{1}{5} \end{bmatrix}$$

1.1.2 Comportement asymptotique des chaînes de Markov

Au cours d'un processus de durée inconnue, il est utile de connaître l'évolution asymptotique de la distribution de probabilité sur les états de la chaîne de Markov, connaissant un état de départ dans la chaîne. En particulier, dans le cas des Processus Décisionnels de Markov, nous verrons que cette distribution de probabilité asymptotique joue un rôle important.

Partant d'une distribution initiale de probabilité p_0, la distribution de probabilité à l'instant t est :

$$p_t = \left(\mathcal{T}^\top\right)^t \cdot p_0 \qquad (\mathcal{T}^\top \text{ est la transposée de } \mathcal{T})$$

Théorème 1 *Distribution stationnaire d'une chaîne de Markov ergodique [45]*

Soit une chaîne de Markov stationnaire de matrice de transition $\mathcal{T}$.
Si la chaîne est ***récurrente*** *(pour chaque état, l'espérance de la durée avant le retour sur cet état est finie), il existe une distribution de probabilité stationnaire p_∞, qui est un vecteur propre associé à la valeur propre 1 de $\mathcal{T}^\top$:*

$$p_\infty = \mathcal{T}^\top \cdot p_\infty$$

Si, de plus, la chaîne est ***irréductible*** *(tout état est accessible à partir de n'importe quel autre état), la distribution stationnaire p_∞ est unique.*
Une chaîne de Markov récurrente et irréductible est dite ***ergodique****.*

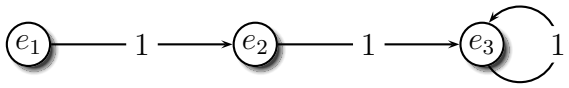


FIG. 1.2 – Chaîne de Markov non ergodique

Sous certaines conditions [45], la suite de distributions $(p_t)_{t\in\mathbb{N}}$ converge vers une probabilité stationnaire p_∞, qui est le point fixe de la matrice $\mathcal{T}^\top$ (cf. théorème 1).

Le théorème ergodique ne donne qu'une condition suffisante de convergence de la distribution de probabilité. Par exemple, la chaîne de Markov de la figure 1.2 n'est pas ergodique, car elle n'est pas récurrente (le temps moyen de premier retour en partant de e_1 ou e_2 est infini). Ainsi, le théorème ergodique ne s'applique pas sur cette chaîne de Markov, dont la distribution stationnaire existe pourtant : $p_\infty = [0\ 0\ 1]$. Cette probabilité est obtenue dès le deuxième instant du processus, quel que soit l'état de départ :

$$p_2 = \left(\mathcal{T}^\top\right)^2 \cdot p_0 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}^2 \cdot p_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \cdot p_0 = \begin{bmatrix} 0 \\ 0 \\ p_0(e_1) + p_0(e_2) + p_0(e_3) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Cette distribution de probabilité est ensuite conservée, car $[0\ 0\ 1]$ est un vecteur propre de $\mathcal{T}^\top$ associé à la valeur propre 1.

Ce calcul prouve qu'un algorithme de point fixe récursif permet de trouver la distribution stationnaire de cette chaîne de Markov, dont l'existence ne peut pas être prouvée par le théorème ergodique. Ainsi que nous l'expliquerons dans la suite, cette idée est utilisée dans les MDP décomposés afin de calculer la distribution de probabilité stationnaire sur les états du MDP en suivant une politique donnée. Précisons enfin qu'il existe un algorithme permettant d'identifier des sous-chaînes de markov ergodiques [46], dans lesquelles il est possible de calculer des sous-distributions de probabilités stationnaires.

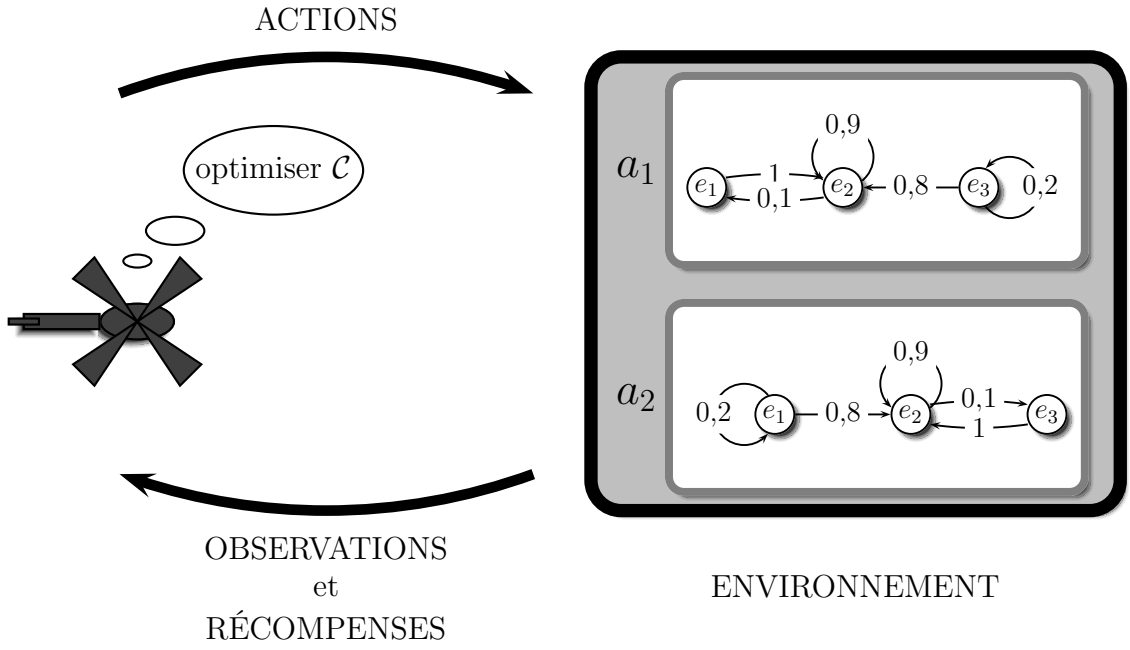


FIG. 1.3 – L'agent autonome contrôle les chaînes de Markov de ses actions en optimisant un critère $\mathcal{C}$ donné.

1.2 Des chaînes de Markov aux Processus Décisionnels de Markov

Considérons un agent autonome qui évolue dans un espace d'états de dimension fini, et dont les effets des actions sont stochastiques et markoviens : la probabilité d'arriver dans un état e' ne dépend que de l'état précédent e et de l'action a appliquée dans l'état e. Ainsi, **pour chaque action, la dynamique de l'agent est une chaîne de Markov**. L'ensemble des chaînes de Markov qui définissent la dynamique des états pour chaque action constitue une chaîne de Markov contrôlée par l'agent. A chaque observation de l'état courant, l'agent décide d'appliquer une action qui implique un changement de chaîne de Markov pour l'ensemble de ses états.

Rajoutons à ce formalisme des récompenses sur les transitions stochastiques entre les états de la chaîne de Markov contrôlée. L'agent peut alors décider des actions optimales à appliquer, de sorte à optimiser un critère numérique basé sur les récompenses de la chaîne. Comme le montre la figure 1.3, l'agent contrôle donc les chaînes de Markov de la dynamique de l'environnement en choisissant dans chaque état, par ses actions, celle qui optimise un critère donné.

Les chaînes de Markov contrôlées, auxquelles des récompenses sont associées aux

transitions stochastiques, sont appelées *Processus Décisionnels de Markov* (MDP). Le formalisme traditionnel des MDP [14], tel que présenté dans la définition 2, repose sur les chaînes de Markov contrôlées, mais sans les nommer. La définition proposée suppose que les transitions et leurs récompenses sont stationnaires, c'est-à-dire indépendantes du temps. De plus, en opposition aux MDP factorisés que nous étudierons dans le chapitre suivant, les MDP traditionnels sont énumérés, au sens où l'espace d'états est entièrement explicité.

Définition 2 Processus Décisionnel de Markov (énuméré)

Un Processus Décisionnel de Markov (stationnaire) est un quadruplet $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ où :

- $\mathcal{E}$ est l'ensemble des états du processus,
- $\mathcal{A}$ est l'ensemble des actions de l'agent,
- $\mathcal{T} = (T^a)_{a \in \mathcal{A}}$ est l'ensemble des transitions stochastiques entre les états pour chaque action :

$$\forall\, a \in \mathcal{A}\ ,\ T^a\ :\ \begin{array}{lcl} \mathcal{E}^2 & \rightarrow & [0;1] \\ (e, e') & \mapsto & P\left(E' = e' \mid a, E = e\right) \end{array}$$

- $\mathcal{R} = (R^a)_{a \in \mathcal{A}}$ est l'ensemble des récompenses associées aux transitions stochastiques :

$$\forall\, a \in \mathcal{A}\ ,\ R^a\ :\ \begin{array}{lcl} \mathcal{E}^2 & \rightarrow & \mathbb{R} \\ (e, e') & \mapsto & r\left(E' = e' \mid a, E = e\right) \end{array}$$

Les fonctions $(T^a)_{a \in \mathcal{A}}$ et $(R^a)_{a \in \mathcal{A}}$ correspondent à des matrices, qui sont généralement appelées respectivement *matrices des transitions* et *matrices des récompenses* du MDP. Ces matrices sont souvent creuses dans la mesure où elles contiennent une majorité d'éléments nuls car, partant d'une état donné, les états successeurs possibles pour une action quelconque sont en général très peu nombreux. Par exemple, dans le cas d'un espace d'états géographiques, les états ne sont connectés qu'à leurs voisins topologiques, dont le nombre est faible comparé à l'espace d'états total.

Aussi, les matrices de transition et de récompenses sont généralement implémentées sous forme de matrices creuses [47, 48, 49], qui offrent dans la plupart des cas des performances nettement supérieures aux matrices pleines, où tous les éléments sont enregistrés en mémoire, y compris ceux qui sont nuls. Néanmoins, l'utilisation des matrices creuses peut s'avérer désastreuse pour des MDP denses, c'est-à-dire dont le nombre de transitions de probabilités non nulles est de l'ordre de $|\mathcal{E}|^2$ pour chaque action [50, 21].

1.3 Optimisation de la stratégie

1.3.1 Critères d'optimisation

Les but des MDP est de produire une *politique* optimale, c'est-à-dire une fonction π qui associe à chaque état une action optimale au regard d'un critère C donné. En général, ce critère s'appelle *fonction de valeur* car, pour chaque état e, il définit une valeur moyenne reçue par l'agent au cours du processus s'il part initialement de l'état e. Les principaux critères d'optimisation sont :

- critère fini : $\mathcal{C}(e) = E\left[\sum_{t=0}^{T} r_t \mid \pi, e_0 = e\right]$, où T est l'*horizon* du processus (nombre d'instants) ;
- critère γ-pondéré : $\mathcal{C}(e) = E\left[\sum_{t=0}^{+\infty} \gamma^t r_t \mid \pi, e_0 = e\right]$, où γ est le facteur d'actualisation ($0 \leqslant \gamma < 1$) ;
- critère total : $\mathcal{C}(e) = E\left[\sum_{t=0}^{+\infty} r_t \mid \pi, e_0 = e\right]$
- critère moyen : $\mathcal{C}(e) = E\left[\lim_{T\to+\infty} \frac{1}{T}\sum_{t=0}^{T} r_t \mid \pi, e_0 = e\right]$

Le critère fini présente un intérêt lorsque la longueur de la politique optimale (le nombre de coups pour arriver au but) est connue. Par exemple, si nous supposons que la chaîne de Markov de la figure 1.2 définit un MDP à une seule action, l'état e_3 est atteignable en 2 coups maximum depuis tous les états de la chaîne.

Néanmoins, si les états sont récurrents, une politique peut produire des cycles dans la chaîne de Markov au point que le nombre de coups pour arriver au but est inconnu. Aussi, dans ce cas, les critères à horizon infinis sont plus intéressants car ils optimisent la politique sur la base d'un nombre de coups inconnu. Notons toutefois que la politique obtenue n'est pas optimale si le nombre de coups maximum est borné. En effet, une fois ce nombre de coups dépassé, la variation du critère n'est pas nécessairement monotone.

L'inconvénient des critères moyen et total est que leur série converge seulement dans certains cas particuliers [14]. Ainsi, en horizon infini, le critère γ-pondéré est largement utilisé car le coefficient d'actualisation γ assure une convergence de la série. Ce coefficient, dont la raison d'être n'est a priori que mathématique, peut être interprété comme une probabilité de survie de l'agent ($1 - \gamma$ représente alors la probabilité d'arrêt du processus). Cette interprétation est toutefois limitée car elle implique que la probabilité de panne du système coûte 0, ce qui n'est pas très réaliste. Dans toute la suite, nous ne considérerons plus que le critère γ-pondéré.

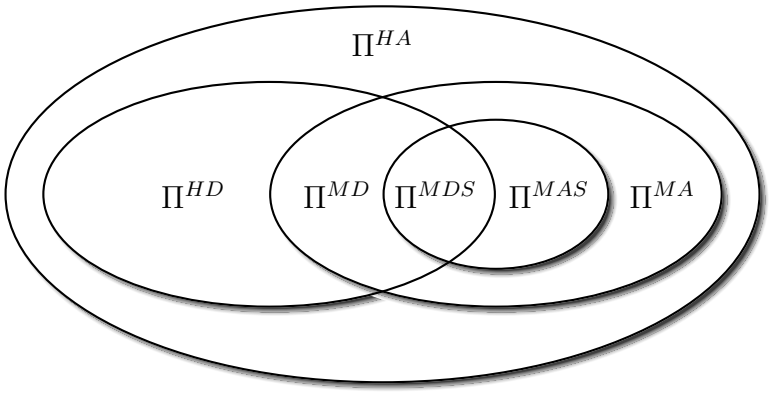


FIG. 1.4 – Relations ensemblistes entre les différentes catégories de politiques

1.3.2 Caractérisation des politiques optimales

Les politiques optimales sont, par définition, celles qui maximisent (ou minimisent) le critère d'optimisation $\mathcal{C}$. Dans le cas général, cette politique est histoire-dépendante aléatoire, c'est-à-dire :

- aléatoire : dans chaque état, l'action à appliquer est tirée au hasard selon une loi de probabilité propre à la politique,
- histoire-dépendante : le choix de chaque action dépend de tout l'historique des états de la chaîne de Markov définie par la politique.

La dynamique de l'agent et celle de sa politique d'actions sont différentes, si bien que la politique peut être non markovienne alors que la dynamique de l'agent l'est. En notant $\mathcal{F}(X, Y)$ l'ensemble des fonctions de X dans Y, $\mathcal{P}(X)$ l'ensemble des parties de X, et $\mathcal{L}(Y)$ une loi de probabilité sur Y, les ensembles de politiques seront notés de la manière suivante :

- $\Pi^{HA} = \mathcal{F}(\mathcal{P}(\mathcal{E}) \times \mathbb{N}, \mathcal{L}(\mathcal{A}))$: politiques histoire-dépendante aléatoires ;
- $\Pi^{HD} = \mathcal{F}(\mathcal{P}(\mathcal{E}) \times \mathbb{N}, \mathcal{A})$: politiques histoires-dépendantes déterministes ;
- $\Pi^{MA} = \mathcal{F}(\mathcal{E} \times \mathbb{N}, \mathcal{L}(\mathcal{A}))$: politiques markoviennes aléatoires ;
- $\Pi^{MD} = \mathcal{F}(\mathcal{E} \times \mathbb{N}, \mathcal{A})$: politiques markoviennes déterministes.
- $\Pi^{MAS} = \mathcal{F}(\mathcal{E}, \mathcal{L}(\mathcal{A}))$: politiques markoviennes aléatoires stationnaires ;
- $\Pi^{MDS} = \mathcal{F}(\mathcal{E}, \mathcal{A})$: politiques markoviennes déterministes stationnaires.

Il est tout à fait possible qu'une politique optimale d'un MDP stationnaire soit elle-même non stationnaire : par exemple, en horizon fini avec un but atteignable en $T > 1$ coups maximum, la politique en 1 coup est inepte dans certains états alors que celle en T coups mène au but depuis tous les états. La figure 1.4 représente les relations ensemblistes entre les différentes catégories de politiques.

Comme l'indique la définition 3, les politiques optimales maximisent le critère d'optimisation sur l'ensemble de politiques le plus général. Pour une politique markovienne déterministe π, sa fonction de valeur V_γ^π peut être calculée très simplement

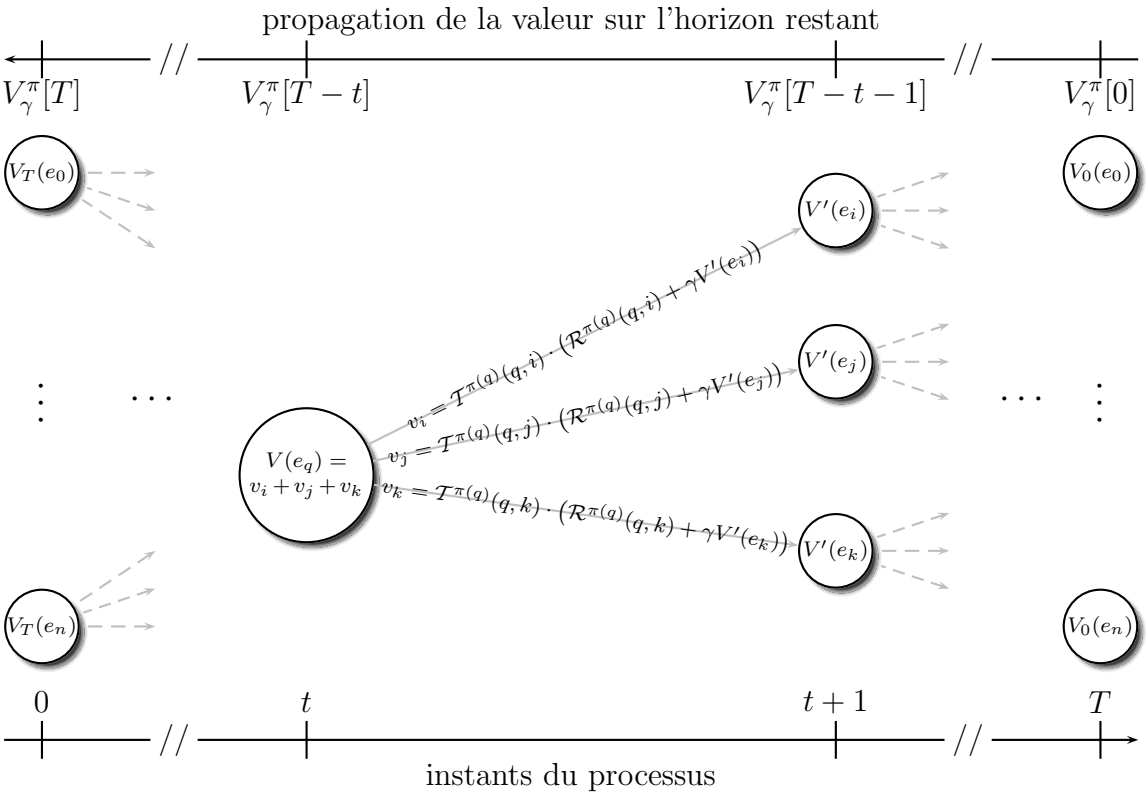


FIG. 1.5 – Principe de la programmation dynamique stochastique

par programmation dynamique stochastique [15, 38] (cf. figure 1.5) : la valeur d'un état e_q suite à la réalisation de l'action $\pi(e_q)$, lorsqu'il reste $T-t$ coups, est mise à jour en ajoutant la moyenne des récompenses directes à la moyenne des valeurs des états successeurs, pour lesquels il reste $T-t-1$ coups.

Définition 3 Politiques optimales

Soit un MDP $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ et un coefficient d'actualisation $0 \leqslant \gamma < 1$.
L'ensemble Π^* des politiques optimales est défini par :

$$\Pi^* = \left\{ \pi^* \in \Pi^{HA} \ : \ \forall\, \overline{e} \in \mathcal{P}(\mathcal{E}) \ , \ \pi^*(\overline{e}) \in \underset{\pi \in \Pi^{HA}}{\operatorname{argmax}} \underbrace{E\left[\sum_{t=0}^{+\infty} \gamma^t r_t \mid \pi, e_0 = e \right]}_{V_\gamma^\pi} \right\}$$

V_γ^π est la *fonction de valeur* de la politique π.

Aussi, en augmentant l'horizon du processus jusqu'à l'infini, il est possible d'obtenir une caractérisation de la fonction de valeur des politiques markoviennes déterministes (cf. théorème 2). La moyenne des gains cumulées converge, car le coefficient d'actualisation γ est strictement inférieur à 1.

Théorème 2 *Caractérisation de la fonction de valeur d'une politique markovienne déterministe stationnaire [14]*

Soit un MDP $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *et un coefficient d'actualisation* $0 \leqslant \gamma < 1$.
Soit une politique markovienne déterministe π .
La valeur V_γ^π *de la politique* π *existe, et est l'unique solution de l'équation :*

$$\forall\, e \in \mathcal{E}\ ,\ V(e) = \underbrace{\sum_{e' \in \mathcal{E}} T^{\pi(e)}(e, e') \cdot \left(R^{\pi(e)}(e, e') + \gamma V(e') \right)}_{L^\pi(V)(e)}$$

L^π *est appelé* opérateur de Bellman. *L'unique point fixe de* L^π *est la limite des suites* $(u_t)_{t \in \mathbb{N}}$ *définies par :*
– $u_0 \in \mathbb{R}^{|\mathcal{E}|}$
– $\forall\, t > 0\ ,\ u_{t+1} = L^\pi(u_t)$

Le théorème 2, combiné à la définition 3, permet de calculer les politiques markoviennes déterministes par programmation dynamique. Or, il est prouvé [14] que pour toute politique histoire-dépendante aléatoire optimale, il existe toujours une politique markovienne déterministe stationnaire de même fonction de valeur. Ainsi, il suffit de chercher les politiques optimales parmi les politiques markoviennes déterministes stationnaires, dont la caractérisation est donnée dans le théorème 2. Ce résultat est l'objet du théorème fondamental 3, qui caractérise les politiques optimales à l'aide de l'équation de Bellman, définie sur l'ensemble des politiques markoviennes déterministes stationnaires.

La figure 3 (page 11) donne un exemple de politique markovienne déterministe stationnaire. Ce théorème fondamental est à l'origine de deux classes d'algorithmes d'optimisation des MDP qui recherchent les politiques optimales dans le sous-espace des politiques markoviennes déterministes stationnaires, en itérant jusqu'à trouver le point fixe de l'opérateur de Bellman. Ces deux classes d'algorithme sont appelées *itération de la valeur* et *itération de la politique*, suivant que les algorithmes itèrent sur la fonction de valeur ou les politiques elles-mêmes.

Précisons enfin qu'il existe des théorèmes de caractérisation des fonction de valeur et des politiques optimales pour chaque critère d'optimisation [14]. Comme nous l'avons déjà mentionné, ces théorèmes n'assurent pas la contraction des points fixes des opérateurs de Bellman correspondant aux critères d'horizon infinis autres que le critère γ-pondéré. Aussi, parmi les critères d'horizon infini, seul le critère γ-pondéré

Théorème 3 *Caractérisation des politiques optimales [14]*

Soit un MDP $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *et un coefficient d'actualisation* $0 \leqslant \gamma < 1$.

1. *Pour toute politique optimale, il existe une politique markovienne déterministe stationnaire de même fonction de valeur :*

$$\Pi^* \supset \left\{ \pi^* \in \Pi^{MDS} \ : \ \forall \ e \in \mathcal{E} \ , \ \pi^*(e) \in \underset{\pi \in \Pi^{MDS}}{\operatorname{argmax}} V_\gamma^\pi(e) \right\}$$

2. *Les fonctions de valeurs* $V_\gamma^{\pi^*}$ *des politiques markoviennes déterministes stationnaires optimales* π^* *sont l'unique solution de l'*équation de Bellman *:*

$$\forall \ e \in \mathcal{E} \ , \ V_\gamma^*(e) = \underbrace{\max_{a \in \mathcal{A}} \left\{ \sum_{e' \in \mathcal{E}} T^a(e, e') \cdot \left(R^a(e, e') + \gamma V_\gamma^*(e') \right) \right\}}_{L(V_\gamma^*)(e')}$$

   L'unique point fixe de l'opérateur de Bellman L *est la limite des suites* $(u_t)_{t \in \mathbb{N}}$ *définies par :*
   - $u_0 \in \mathbb{R}^{|\mathcal{E}|}$
   - $\forall \ t > 0 \ , \ u_{t+1} = L(u_t)$

3. *Est optimale toute politique stationnaire* π^* *définie par :*

$$\pi^* \in \underset{\pi \in \Pi^{MDS}}{\operatorname{argmax}} \left\{ L^\pi \cdot V_\gamma^* \right\}$$

est utilisable en pratique lorsque la structure des chaînes de Markov du MDP sont inconnues a priori.

1.3.3 Algorithmes d'optimisation des MDP

Itération de la valeur

Comme son nom l'indique, l'algorithme 1 d'itération de la valeur calcule itérativement le point fixe de l'opérateur de Bellman L jusqu'à la convergence du critère d'optimisation (cf. théorème 3). Toutes les normes étant équivalentes en dimensions finies, il est possible de choisir n'importe quelle norme sur $\mathcal{E}$. Toutefois, pour le critère γ-pondéré, la norme infinie est généralement utilisée :

$$\forall \ V \in \mathbb{R}^{|\mathcal{E}|} \ , \ \|V\|_\infty = \max_{e \in \mathcal{E}} |V(e)|$$

Algorithme 1 : Fonction `IterationValeur` [14]

Données : $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, $0 \leqslant \gamma < 1$, $\epsilon > 0$
Résultat : π^*, V_γ^*
début
  $V \leftarrow 0$;
  répéter
    $V' \leftarrow V$;
    pour $e \in \mathcal{E}$ **faire**
      pour $a \in \mathcal{A}$ **faire**
        $somme \leftarrow 0$;
        pour $e' \in \mathcal{E}$ **faire**
          $somme \leftarrow somme + (T^a(e, e') \cdot (R^a(e, e') + \gamma V'(e')))$;
        si $a = \mathcal{A}.premier()$ **alors**
          $V(e) \leftarrow somme$;
        sinon
          $V(e) \leftarrow \max(V(e), somme)$;
  jusqu'à $\|V - V'\|_\infty < \epsilon$;
  pour $e \in \mathcal{E}$ **faire**
    pour $a \in \mathcal{A}$ **faire**
      $somme \leftarrow 0$;
      pour $e' \in \mathcal{E}$ **faire**
        $somme \leftarrow somme + (T^a(e, e') \cdot (R^a(e, e') + \gamma V(e')))$;
      si $a = \mathcal{A}.premier()$ **ou** $somme > V(e)$ **alors**
        $\pi(e) \leftarrow a$;
  retourner (π, V);
fin

Dès que la fonction de valeur des politiques optimales a convergé ϵ près, une politique optimale markovienne déterministe stationnaire est calculée en 1 coup de programmation dynamique (cf. théorème 3). Grâce à la stricte contraction de l'opérateur de Bellman, il est possible d'estimer un majorant du nombre d'itérations suffisant pour obtenir une politique ϵ-optimale [14] :

$$\forall\, t \in \mathbb{N}\ ,\ \|V_\gamma^t - V_\gamma^*\| \leqslant \frac{\gamma^t}{1-\gamma}\|V_\gamma^1 - V_\gamma^0\|$$

Dès que le membre de droite de l'inégalité précédente est inférieur à $\frac{\epsilon}{2}$, l'algorithme

cesse d'itérer sur la valeur :

$$\frac{\gamma^t}{1-\gamma}\|V_\gamma^1 - V_\gamma^0\| < \frac{\epsilon}{2} \quad \Longrightarrow \quad \|V_\gamma^{t+1} - V_\gamma^t\| < \epsilon$$

Par conséquent, un majorant T_{max} du nombre d'itérations suffisant, pour que la fonction de valeur soit optimale à ϵ près, est[1] :

$$T_{max} = E_{\mathbb{N}}\left(\frac{\ln \epsilon(1-\gamma) - \ln 2\|V^1 - V^0\|}{\ln \gamma}\right) + 1$$

où $E_{\mathbb{N}}$ est la fonction "partie entière".
La complexité de l'algorithme d'itération de la valeur est donc [18] :

$$\boxed{O\left(\frac{\ln \epsilon(1-\gamma)}{\ln \gamma}|\mathcal{A}||\mathcal{E}|^2\right)}$$

Ainsi, il apparaît que la complexité de l'algorithme augmente quand la précision diminue et que le facteur d'actualisation se rapproche de 1. En particulier, dans le cas limite $\gamma = 1$ (critère total), le nombre d'itérations peut être infini. Autrement dit, l'algorithme d'itération de la valeur ne converge pas nécessairement pour le critère total.

Plusieurs versions de l'algorithme d'itération de la valeur existent [14], si bien que la fonction `IterationValeur` constitue le squelette d'une classe d'algorithmes d'itération de la valeur. Entre autres variantes, la version de Gauss-Seidel consiste à mettre à jour la fonction de valeur sauvegardée V' au fur et à mesure que la nouvelle fonction de valeur V est calculée, de sorte à accélérer la convergence du point fixe de l'opérateur de Bellman.

Une autre version s'appuie sur une représentation des transitions sous forme de matrices creuses, qui s'avère très efficace lorsque la densité en transitions pour chaque action est négligeable devant $|\mathcal{E}|^2$ [21]. Une telle représentation est strictement équivalente à un graphe de transitions où les arêtes de probabilités nulles ne sont pas enregistrées. Ainsi, la somme sur tous les états e' de $\mathcal{E}$ se réduit avantageusement à la somme sur les nœuds successeurs dans le graphe de transitions [21]. Aussi, nous avons développé une bibliothèque d'optimisation de MDP énumérés définis sous forme de graphe de transitions, appelée `graphMDP`, qui prend en compte plusieurs critères d'optimisation (cf. annexe B page 275). Nous verrons dans le chapitre suivant qu'une telle représentation est particulièrement intéressante pour la réduction de l'espace d'états par des techniques de partitionnement de graphes.

D'autre part, le formalisme classique des MDP suppose que chaque action est applicable dans tous les états, ce qui n'est pas le cas. En particulier, lorsqu'une

[1] Connaissant T_{max}, nous pourrions être tentés d'optimiser le MDP sur un horizon fini. Ceci est une mauvaise idée, car le nombre d'itérations réel est souvent très largement inférieur à T_{max}.

action n'est définie que dans un sous-ensemble très restreint d'états, l'algorithme d'itération de la valeur présenté itère inutilement sur des états où l'action n'est pas applicable. Aussi, une représentation des MDP énumérés sous forme de graphes de transitions (ou de matrices creuses), comme dans `graphMDP`, permet de définir pour chaque état uniquement des arêtes qui correspondent aux actions applicables dans cet état.

Itération de la politique

L'algorithme 2 d'itération de la politique met à jour itérativement une politique markovienne déterministe après l'avoir évaluée à chaque itération. La politique courante π est elle-même évaluée itérativement, en tant que point fixe de l'opérateur de Bellman L^π (cf. théorème 2). Dès que la politique courante est évaluée, une nouvelle politique markovienne déterministe est calculée en 1 coup de programmation dynamique (cf. théorème 3). Le processus itératif s'arrête lorsque la politique est stationnaire, puisque les politiques optimales peuvent être recherchées dans le sous-ensemble des politiques markoviennes déterministes stationnaires (cf. théorème 3).

La complexité de l'algorithme d'itération de la politique est *a priori* supérieure à celle de l'algorithme d'itération de la valeur, car le calcul du point fixe de L est aussi complexe que celui du point fixe de L^π. Ainsi, le nombre de politiques markoviennes déterministes stationnaires possibles étant $|\mathcal{A}|^{|\mathcal{E}|}$, la complexité de l'algorithme d'itération de la politique est majorée par :

$$\boxed{O\left(\frac{\ln \epsilon(1-\gamma)}{\ln \gamma}|\mathcal{A}|^{|\mathcal{E}|}|\mathcal{E}|^2\right)}$$

Néanmoins, l'algorithme d'itération de la politique converge généralement plus vite que l'algorithme d'itération de la valeur, car l'espace des politiques est de dimension finie, contrairement à l'espace des fonctions de valeur. De plus, il est prouvé [14] que la vitesse de convergence de l'algorithme d'itération de la politique peut être encore améliorée en dégradant l'évaluation de la politique courante, sans perte d'optimalité. En comparant les complexités des algorithmes d'itération de la politique et d'itération de la valeur, il est clair qu'une dégradation de la précision ϵ peut avantager l'algorithme d'itération de la politique. D'autre part, dans certains cas particuliers, la complexité peut être encore réduite [14].

Comme pour l'algorithme d'itération de la valeur, l'algorithme d'itération de la politique possède de nombreuses variantes [14]. Dans la version originale, peu efficace, la politique courante est évaluée par inversion du système linéaire : $V_\gamma^\pi = L^\pi \cdot V_\gamma^\pi$. La version présentée dans l'algorithme 2 est appelée *itération de la politique modifiée*, car la politique est évaluée par programmation dynamique, qui peut elle-même suivre le procédé de Gauss-Seidel. D'autres versions proposent également de

Algorithme 2 : Fonction `IterationPolitique`

Données : $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, $0 \leqslant \gamma < 1$, $\epsilon > 0$
Résultat : π^*, V_γ^*
début
  $\pi \leftarrow \mathcal{A}.premier()$;
  répéter
    $V_\pi \leftarrow 0$;
    répéter
      $V'_\pi \leftarrow V_\pi$;
      pour $e \in \mathcal{E}$ **faire**
        $V_\pi(e) \leftarrow 0$;
        pour $e' \in \mathcal{E}$ **faire**
          $V_\pi(e) \leftarrow V_\pi(e) + \left(T^{\pi(e)}(e,e') \cdot \left(R^{\pi(e)}(e,e') + \gamma V'_\pi(e')\right)\right)$;
    jusqu'à $\|V_\pi - V'_\pi\|_\infty < \epsilon$;
    $\pi' \leftarrow \pi$;
    pour $e \in \mathcal{E}$ **faire**
      pour $a \in \mathcal{A}$ **faire**
        $somme \leftarrow 0$;
        pour $e' \in \mathcal{E}$ **faire**
          $somme \leftarrow somme + (T^a(e,e') \cdot (R^a(e,e') + \gamma V_\pi(e')))$;
        si $a = \mathcal{A}.premier()$ **ou** $somme > V^\pi(e)$ **alors**
          $\pi(e) \leftarrow a$;
  jusqu'à $\pi = \pi'$;
  retourner (π, V_π);
fin

remplacer le test d'égalité de la politique courante et de la politique précédente, par un test d'égalité entre les fonctions de valeur de ces politiques.

1.3.4 Vers des modèles structurés de MDP

Une complexité algorithmique polynomiale en le nombre d'états du MDP

La complexité des algorithmes d'itération de la valeur et de la politique est polynomiale en le nombre d'états du MDP. Ainsi, l'utilisation de ces algorithmes dans leur formalisme classique est rédhibitoire pour optimiser des MDP de grande taille. Or, l'espace d'états des problèmes de navigation et de prise d'informations est proportionnel en le nombre d'états géographiques n_g et exponentiel en le nombre d'informations n_i à collecter : $|\mathcal{E}| = n_g 2^{n_i}$. Par conséquent, le nombre d'états aug-

mente très vite avec le nombre d'informations, et plus généralement avec le nombre de caractéristiques (autonomie, vent, vitesse, ...) intégrées à l'espace d'états. Un espace d'états constitué de n caractéristiques $(C_i)_{1 \leqslant i \leqslant n}$ a une cardinalité exponentielle en le nombre de caractéristiques, ce qui est connu sous le nom d'*explosion combinatoire* [18, 16] :

$$|\mathcal{E}| = \prod_{i=1}^{n} |C_i|$$

Espaces d'états structurés

Pourtant, l'espace d'états des problèmes de navigation et de prise d'informations possède une structure particulière dont il serait dommage de ne pas profiter. Premièrement, il est plus facile et plus compact de décrire les états à l'aide de leur caractéristiques plutôt qu'en les énumérant tous. En effet, une *instanciation* partielle des caractéristiques, c'est-à-dire un assignement partiel de valeurs aux caractéristiques, décrit un sous-ensemble d'états de l'espace total qui correspond aux valeurs de toutes les caractéristiques non instanciées. Ainsi, une description basée sur des caractéristiques de l'espace d'états permet une description compacte et groupée des états du MDP.

Deuxièmement, ce principe de regroupement des états, afin de raisonner sur un espace d'états plus compact, peut être appliqué dans certains cas à une description énumérée des états. Lorsque l'espace d'états est structuré en régions faiblement couplées, au sens où le nombre de transitions entre les régions est faible par rapport au nombre de transitions entre les états internes aux régions, il est possible de distribuer l'optimisation du MDP dans les régions de l'espace d'états. L'optimisation dans les régions est beaucoup plus simple que dans le MDP total, car leurs nombres d'états est en général faible devant le nombre total d'états.

Aussi, l'espace d'états des problèmes de déplacement et de prise d'informations possède ces deux types de structure :

- les états sont décrits par des caractéristiques de la mission (position géographique de l'agent, nombre et positions des informations à collecter, autonomie, vitesse, ...) ;
- le sous-espace de navigation est partitionné en régions géographiques naturelles (pièces pour un robot d'intérieur, vallées, espaces urbains, zones boisées, etc. pour un drone).

Des techniques spécifiques, que nous présentons dans le chapitre suivant, ont été développées afin de modéliser et optimiser des MDP dont l'espace d'états est **soit** implicite et décrit par caractéristiques (*factorisé*) [26, 28, 31, 33] **soit** énuméré et partitionné en régions faiblement couplées [22, 23, 24, 25]. A notre connaissance, il n'existe pas de méthode ou de modèle qui prenne en compte, dans une représenta-

tion globalement factorisée, le partitionnement en régions du sous-espace engendré par une caractéristique (position géographique par exemple). Aussi, dans les parties suivantes, nous proposerons un modèle hybride de MDP ainsi qu'un algorithme d'optimisation, où l'espace d'états est globalement factorisé, mais avec une composante partitionnée en régions faiblement couplées. Nous montrerons en particulier l'intérêt et l'efficacité de notre approche pour des problèmes de déplacement et de prise d'informations.

Travaux connexes

Les méthodes hiérarchiques sont souvent utilisées dans les MDP ou en apprentissage par renforcement, afin de déléguer l'optimisation du problème total à des sous-problèmes plus simples. Orthogonalement aux méthodes de partitionnement d'espace d'états, des algorithmes tels `MAXQ` [51] et `HEXQ` [52] ont été proposées pour optimiser le critère de décision sur la base d'actions hiérarchiques, d'abord très générales, puis de plus en plus spécialisées au fur et à mesure du processus d'optimisation.

Par ailleurs, lorsque l'espace d'états est implicitement défini par des caractéristiques, il existe des méthodes qui hiérarchisent l'espace d'états en optimisant le critère de décision sur une liste croissante de caractéristiques [53, 54] : plus le nombre de caractéristiques intégrées au critère augmente, plus le sous-espace d'états qu'elles définissent diminue. D'autres méthodes partitionnent des espaces d'états implicites et factorisés au cours de l'optimisation du MDP, en définissant des partitions et dynamiques non homogènes par rapport au critère de décision [55].

Néanmoins, toutes ces méthodes créent une hiérarchie dynamique et éphémère durant l'optimisation du MDP, alors que les méthodes que nous proposerons dans les parties suivantes visent à construire un modèle de MDP hiérarchique, préalablement à son optimisation, afin d'obtenir un espace d'états réduit et statique, qui facilite ensuite l'optimisation du MDP hiérarchique. Contrairement aux méthodes que nous venons d'évoquer, notre hiérarchisation des états et des actions est obtenue par des techniques de partitionnement de MDP énumérés sur les sous-espaces engendrés par certaines caractéristiques de l'espace d'états.

Chapitre 2

Modèles structurés de MDP

2.1 Diviser pour régner : optimisation de MDP par décomposition en sous-MDP

Considérons un MDP dont l'espace d'états est partitionné en régions faiblement communiquantes, comme dans le figure 2.1. Dans cette section, nous présentons une technique, appelée *décomposition de MDP*, visant à optimiser le MDP dans chaque région de l'espace d'états grâce à des MDP locaux, puis à construire un MDP abstrait dont l'espace d'états est constitué des régions du partitionnement (*macro-états*), et dont les actions sont des macro-déplacements (*macro-actions*) entre les régions. Le partitionnement de l'espace d'états peut être soit donné directement dans le modèle, soit calculé automatiquement par des techniques de partitionnement de graphes [25], qui sont d'autant plus efficaces que les MDP est défini directement sous la forme d'un graphe de transitions. Nous reviendrons sur ces techniques de partitionnement automatique dans la partie sur la réduction automatique du nombre de caractéristiques d'un MDP dont l'espace d'états est factorisé.

Les sous-espaces d'états des régions sont plus petits que l'espace d'états total, si bien que l'optimisation des MDP dans ces régions est plus simple que celle du MDP total. De plus, le MDP abstrait obtenu (cf. figure 2.3) est de taille réduite. Cependant, plus le nombre d'états communiquants entre les régions est grand (4 dans l'exemple de la figure 2.1), plus le nombre de macro-actions est important : en effet, si une région possède q états de sortie, le nombre total de façons de sortir de la région est 2^q. Il est donc primordial que les régions soient faiblement communiquantes.

De manière générale, le nombre maximum de macro-actions partant d'une région r est égal au produit entre le nombre de sorties possibles (ou aucune) et le nombre de politiques interne à la région (*politiques locales*), soit $2^{q+1}|\mathcal{A}|^r$. Ainsi, le gain de réduction de l'espace d'états peut être perdu au détriment d'une augmentation sensible du nombre de macro-actions. La complexité des algorithmes d'itération de

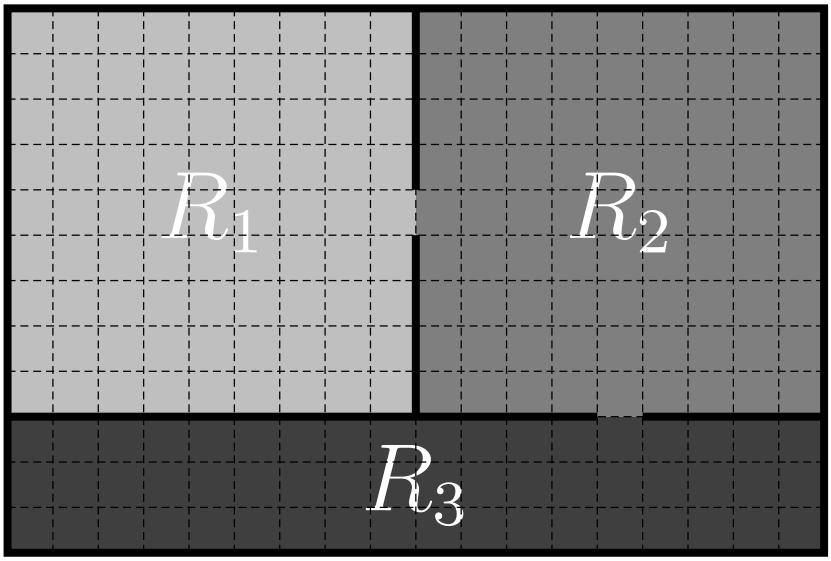


FIG. 2.1 – Espace d'états géographiques constitué de régions faiblement communiquantes R_1, R_2 et R_3

la valeur et d'itération de la politique étant polynomiale en le nombre d'actions, l'optimisation du MDP abstrait risque de devenir à son tour impossible en temps raisonnable.

Heureusement, il est inutile de générer toutes les macro-actions car, pour une façon donnée de sortir de la région, toutes les macro-actions ne permettent pas à l'agent d'acquérir en moyenne les mêmes récompenses accumulées. Ainsi, pour chaque combinaison de sorties, il suffit de générer une macro-action qui maximise les récompenses accumulées pour cette configuration de sortie. De plus, une même macro-action peut maximiser des combinaisons de sorties différentes, ce qui réduit encore le nombre de macro-actions à générer (les déplacements sont stochastiques si bien qu'une même macro-action peut conduire à des sorties différentes). Ainsi, après avoir défini les notions de MDP abstrait et de MDP locaux [22, 23], nous présenterons une technique permettant de construire un ensemble exhaustif et minimum de macro-actions [24].

2.1.1 Partitionnement de l'espace d'états

La décomposition de MDP suppose la donnée d'un partitionnement Π de l'espace d'états m régions, c'est-à-dire que ces régions englobent tout l'espace d'états, qu'elles

sont non vides et distinctes :

$$\mathcal{R} = \{R_i\}_{1 \leqslant i \leqslant m} \quad \text{tel que} \quad \begin{cases} \forall\ 1 \leqslant i \leqslant m\ ,\ R_i \neq \emptyset \\ \forall\ 1 \leqslant i \neq j \leqslant m\ ,\ R_i \cap R_j = \emptyset \\ \mathcal{E} = \bigcup_{i=1}^{m} R_i \end{cases}$$

Pour chaque région R_i, il est possible d'identifier les *états périphériques* de la région, c'est-à-dire ceux internes à la région par lesquels l'agent y rentre, et ceux des autres régions par lesquels l'agent en sort. Les ensembles d'états d'entrée et de sortie de R_i sont notés respectivement $Eper(R_i)$ et $Sper(R_i)$:

- $Eper(R_i) = \{e \in R_i\ :\ \exists\ e' \in \mathcal{E} \setminus R_i\ ,\ \exists\ a \in \mathcal{A}\ ,\ T^a(e', e) \neq 0\}$
- $Sper(R_i) = \{e \in \mathcal{E} \setminus R_i\ :\ \exists\ e' \in R_i\ ,\ \exists\ a \in \mathcal{A}\ ,\ T^a(e, e') \neq 0\}$

Les états périphériques jouent un rôle privilégié dans la décomposition de MDP, car ce sont eux qui permettent la communication entre les régions. Comme nous l'avons évoqué plus haut, la génération des macro-actions sera d'autant plus efficace que les périphéries entrantes et sortantes des régions sont petites. Remarquons que les états entrants et sortants coïncident sur l'ensemble des régions du partitionnement :

$$\bigcup_{i=1}^{m} Eper(R_i) = \bigcup_{i=1}^{m} Sper(R_i)$$

2.1.2 MDP abstrait

Deux modèles de MDP abstraits ont été proposés, suivant que les macro-états sont les états périphériques [23], ou les régions du partitionnement [22] (cf. figure 2.2). Cependant, le premier modèle ne permet pas de définir des macro-actions réentrantes, qui consisteraient par exemple à récolter une récompense de très forte valeur à l'intérieur d'une région donnée : en effet, ce modèle n'autorise que des transitions entre les états entrants et sortants des régions. Une solution serait de rajouter un état absorbant pour chaque région. Néanmoins, le MDP abstrait obtenu avec ce modèle contient bien plus d'états que le MDP abstrait du deuxième modèle. De plus, les états entrants des régions ne nous semblent pas pertinents au regard des stratégies macroscopiques, car l'agent peut partir de n'importe quel état interne d'une région.

Ainsi, nous privilégions le modèle de MDP abstrait dont les macro-états sont les régions du partitionnement [22]. Nous verrons toutefois dans le paragraphe suivant que les états de sortie des régions doivent être conservés afin de calculer les transitions entre les macro-états. Ce modèle est utilisé dans [22] afin de générer des macro-actions optimales pour la stratégie globale, au fur et à mesure de l'optimisation du MDP abstrait : la stratégie optimale est calculée de manière itérative, en alternant une phase d'optimisation du MDP abstrait avec les macro-actions courantes, et une

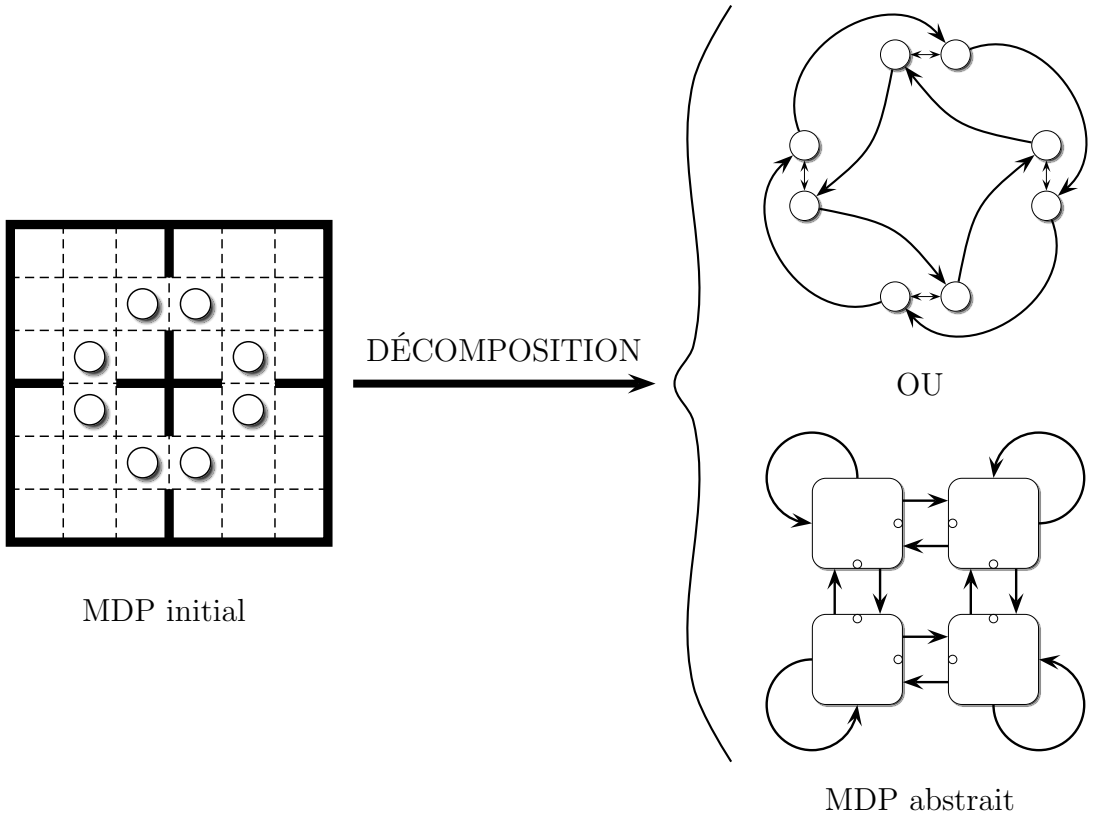


FIG. 2.2 – Deux modèles de MDP abstraits obtenus par décomposition d'un MDP partitionné en 4 régions et 9 états communiquants. Les transitions symboliques indiquent les transitions possibles entre les macro-états.

phase de mise à jour de l'ensemble courant des macro-actions dans le but d'améliorer le critère de la stratégie globale.

Néanmoins, dans notre approche hybride que nous présenterons dans les parties suivantes, nous n'avons pas besoin d'optimiser le MDP abstrait. Nous souhaitons simplement générer le sous-MDP abstrait du sous-espace d'états engendré par une caractéristique donnée d'un MDP factorisé, afin d'optimiser ce dernier (et non le sous-MDP abstrait). Par conséquent, la méthode de construction incrémentale du MDP abstrait [22] ne peut pas être appliquée à notre approche. Aussi, bien que nous exploitons le modèle de MDP abstrait de [22], nous utiliserons la méthode de [23], qui propose de définir le MDP abstrait préalablement à son optimisation, sur la base d'un ensemble de macro-actions données (cf. definition 4). Les macro-actions sont des stratégies définies au niveau de chaque région, si bien qu'elles sont appelées *politiques locales*. Le MDP abstrait ainsi défini est une *décomposition* du MDP initial. Par exemple, le MDP de la figure 2.3 est un MDP abstrait du MDP décrit dans la

figure 2.1, défini par un ensemble donné de politiques locales $\cup_{1 \leqslant i \leqslant m} \left\{\pi_1^i, \ldots, \pi_{m_i}^i\right\}$.

Définition 4 MDP décomposé abstrait

Soit $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ un MDP et $\mathcal{R}$ un partitionnement de son espace d'états $\mathcal{E}$.
Soit $\cup_{r \in \mathcal{R}} \left\{\pi_1^r, \ldots, \pi_{m_r}^r\right\}$ un ensemble donné de politiques locales définies dans chaque région du partitionnement $\mathcal{R}$.
Le MDP décomposé abstrait $\left\langle \tilde{\mathcal{E}}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \right\rangle$ est défini par :

- $\tilde{\mathcal{E}} = \mathcal{R}$;
- $\tilde{\mathcal{A}} = \cup_{r \in \mathcal{R}} \left\{\pi_1^r, \ldots, \pi_{m_r}^r\right\}$;
- $\tilde{\mathcal{T}} = \left(\tilde{T}^\pi\right)_{\pi \in \tilde{\mathcal{A}}}$ tel que :

$$\forall\, \pi \in \tilde{\mathcal{A}}\ , \forall\, (r, r') \in \mathcal{R}^2\ ,\ \tilde{T}^\pi(r, r') = P(r' \mid \pi, r)$$

- $\tilde{\mathcal{R}} = \left(\tilde{R}^\pi\right)_{\pi \in \tilde{\mathcal{A}}}$ tel que :

$$\forall\, \pi \in \tilde{\mathcal{A}}\ , \forall\, (r, r') \in \mathcal{R}^2\ ,\ \tilde{R}^\pi(r, r') = E\left[\sum_{t=0}^{+\infty} \gamma^t r_t \mid \pi, e_0 \in r, e_\infty \in r'\right]$$

Chaque macro-transition $\tilde{T}^\pi(r, r')$ est la probabilité agrégée d'arriver dans un état d'entrée quelconque d'une région r', en partant d'un état quelconque d'une région r et en suivant la politique π. La macro-récompense associée $\tilde{R}^\pi(r, r')$ est la moyenne des récompenses accumulées sur tous les chemins qui partent d'un état quelconque de r et qui arrivent dans un état d'entrée quelconque de r' en suivant la politique π. La politique locale π se termine dès que l'agent arrive dans r'. Ainsi, dans l'expression de la macro-récompense donnée dans la définition 4, l'état e_∞ est un état stationnaire artificiel égal à l'état d'entrée dans la région r'.

2.1.3 Macro-transitions

A priori, les macro-transitions $(\tilde{T}^\pi)_{\pi \in \tilde{\mathcal{A}}}$, qui sont des comportements asymptotiques des politiques locales, ne convergent pas nécessairement. En effet, l'incertitude des actions introduit éventuellement des cycles dans les trajectoires qui partent d'un état d'une région donnée, si bien que les chemins menant à une autre région sont possiblement de longueur infinie. En revanche, les macro-récompenses associées $(\tilde{R}^\pi)_{\pi \in \tilde{\mathcal{A}}}$ convergent grâce au coefficient strictement contracteur γ.

La preuve mathématique de convergence des macro-transitions n'est pas donnée dans [22]. Toutefois, une macro-transition $\tilde{T}^\pi(r, r')$ est une moyenne sur $Sper(r) \cap r'$

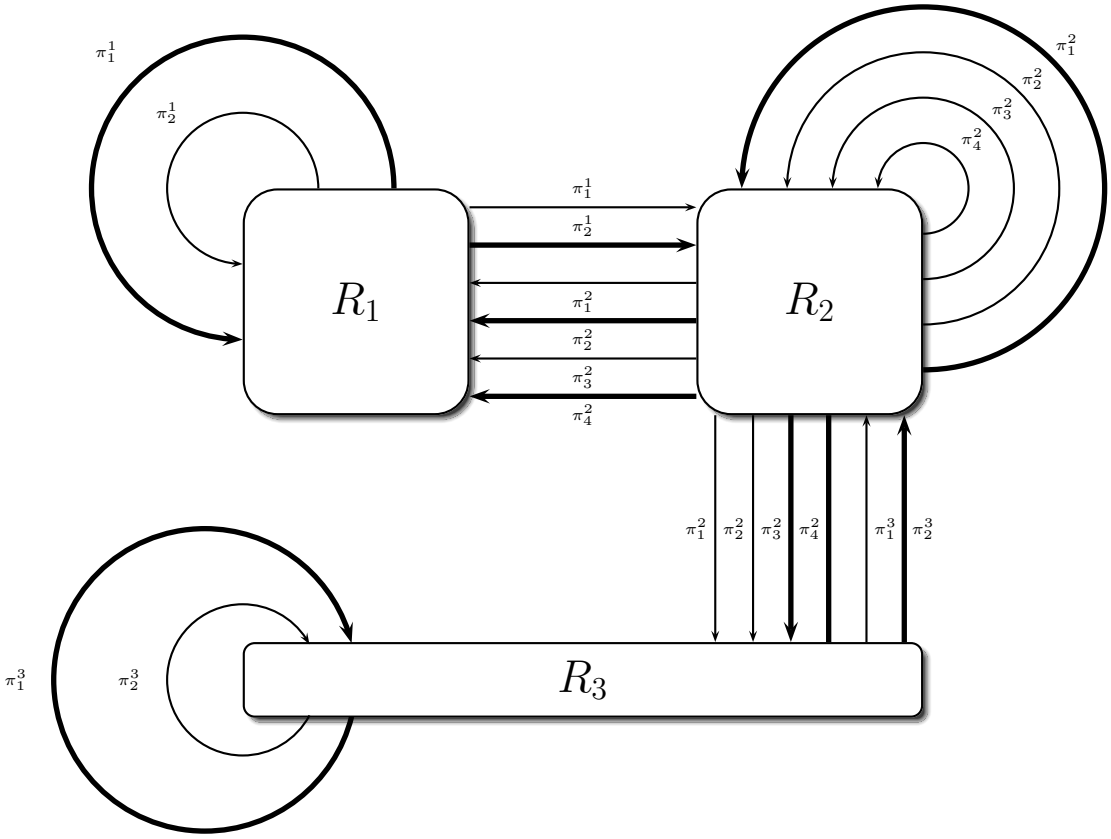


FIG. 2.3 – Exemple de MDP abstrait obtenu par décomposition du MDP de la figure 2.1. Chaque politique locale π_j^i est la $j^{\text{ème}}$ macro-action générée dans la région R_i.

des probabilités stationnaires de la chaîne de Markov définie par les états de $r \cup Eper(r')$ et les actions de la politique π, initialisée par une distribution de probabilité homogène sur r. Or, si cette chaîne de Markov est ergodique (cf. théorème 1 page 15), nous avons vu que la distribution de probabilité stationnaire existe.

Cependant, il est possible de démontrer que les macro-transitions convergent sans aucune hypothèse supplémentaire sur la chaîne de Markov définie par les états de $r \cup Eper(r')$ et la politique π, en utilisant le fait que la politique locale s'arrête dès que l'agent sort de la région. La preuve de ce résultat, non triviale en raison des cycles infinis possibles dans la région de départ, est *exactement la même* que celle que nous proposons pour le calcul de la politique qui maximise la probabilité de rejoindre un ensemble de buts donné en partant d'un ensemble d'états initiaux possibles (cf. chapitre 8 page 195 et théorème 11 page 201) : les états initiaux sont

ceux de la région de départ, et les états buts sont ceux de la région d'arrivée. Notre démonstration repose sur l'extraction d'une sous-matrice inversible de la chaîne de Markov sur l'ensemble des états de la région de départ qui mènent à la région d'arrivée avec une probabilité non nulle (cf. théorème 11 page 201).

Comme l'indique le théorème 4, les macro-transitions et macro-récompenses sont des points fixes d'équations de programmation dynamique spécifiques, qui peuvent être calculés itérativement en partant de valeurs initiales non nulles sur les états de la région d'arrivée (programmation dynamique «arrière»). Pour chaque état de sortie e', la fonction caractéristique $\chi_{\{e'\}}$ joue le rôle de condition d'arrêt : la valeur est propagée sur l'état courant e que si l'état successeur e'' de la programmation dynamique est différent de e' ($\chi_{\{e'\}}(e'') = 0$ et $\chi_r(e'') = 1$). Enfin, la normalisation par $|r|$ signifie que la distribution de probabilité sur l'état de départ de la région initiale est homogène : nous ne disposons a priori d'aucune information sur l'état de départ de l'agent à chaque fois qu'il commence une nouvelle politique locale.

Il peut paraître absurde de calculer les macro-transitions et macro-récompenses dans le but de simplifier ensuite l'optimisation du MDP total puisque, dans les deux cas, des équations de programmation dynamique sont résolues. Néanmoins, nous rappelons que la complexité de la programmation dynamique est polynomiale en le nombre d'états sur lesquels la valeur est propagée. Or, les équations de programmation dynamique qui permettent de calculer les macro-transitions et macro-récompenses itèrent uniquement sur les états internes des régions, qui sont en général bien plus petites que l'espace d'états total. Ainsi, le calcul des macro-transitions et des macro-récompenses présente un intérêt si le MDP initial est de trop grande taille au point qu'il ne peut pas être optimisé. C'est d'ailleurs précisément dans ce but que nous proposerons d'utiliser, dans la partie suivante, des techniques de décomposition de MDP sur les sous-espaces engendrés par certaines caractéristiques d'un MDP factorisé, dont l'optimisation directe est rédhibitoire.

Si l'optimisation du MDP initial est réalisable, il est alors primordial que le temps passé à calculer les macro-transitions et les macro-récompenses pour toutes les politiques n'excède pas le temps d'optimisation du MDP initial. Ainsi, des algorithmes heuristiques de calcul des macro-transitions et des macro-récompenses ont été proposés [56], basés sur une estimation de la distance des états d'une région aux récompenses incluses dans les autres régions. Il est toutefois important que l'ensemble de politiques locales soit le plus petit possible, si bien que nous présentons dans le paragraphe suivant une méthode qui vise à générer un ensemble exhaustif et suffisant (minimum) de politiques locales, sans perte d'optimalité au regard du MDP global [24].

Théorème 4 *Macro-transition et macro-récompense d'une politique locale*

Soient un MDP $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *et un partitionnement* $\mathcal{R}$ *de l'espace d'états* $\mathcal{E}$. *Soient* $r \in \mathcal{R}$ *une région de départ,* $r' \in \mathcal{R}$ *une région d'arrivée, et une politique locale* $\pi \in \mathcal{F}(r, \mathcal{A})$ *définie sur la région* r.
Soit χ_E *la fonction indicatrice d'un sous-ensemble d'états* $E \subset \mathcal{E}$.

– *La macro-transition* $\tilde{T}^\pi(r, r')$ *existe et vaut :*

$$\tilde{T}^\pi(r, r') = \frac{1}{|r|} \sum_{\substack{e \in r \\ e' \in Sper(r) \cap r'}} \overline{T}^\pi(e, e')$$

où, pour tout e' *dans* $Sper(r) \cap r'$, *le vecteur* $\left(\overline{T}^\pi(e, e')\right)_{e \in r}$ *est l'unique point fixe de l'équation de programmation dynamique :*

$$\forall\, e \in r\ ,\ \overline{T}^\pi(e, e') = \sum_{e'' \in r \cup \{e'\}} T^{\pi(e)}(e, e'') \cdot \left(\chi_{\{e'\}}(e'') + \chi_r(e'') \cdot \overline{T}^\pi(e'', e')\right)$$

– *La macro-récompense* $\tilde{\mathcal{R}}^\pi(r, r')$ *existe et vaut :*

$$\tilde{R}^\pi(r, r') = \frac{1}{|r|} \sum_{\substack{e \in r \\ e' \in Sper(r) \cap r'}} \overline{R}^\pi(e, e')$$

où, pour tout e' *dans* $Sper(r) \cap r'$, *le vecteur* $\left(\overline{R}^\pi(e, e')\right)_{e \in r}$ *est l'unique point fixe de l'équation de programmation dynamique :*

$$\forall\, e \in r\ ,\ \overline{R}^\pi(e, e') = \sum_{\substack{e'' \in r \cup \{e'\} \\ \overline{T}^\pi(e'', e') \neq 0}} T^{\pi(e)}(e, e'') \cdot \left(R^{\pi(e)}(e, e'') + \gamma \cdot \chi_r(e'') \cdot \overline{R}^\pi(e'', e')\right)$$

2.1.4 Génération des politiques locales avec des MDP locaux

MDP locaux génériques

Les politiques optimales π^* du MDP global peuvent être décomposées suivant chaque région du partitionnement :

$$\forall\, \pi^* \in \mathcal{F}(\mathcal{E}, \mathcal{A})\ , \forall\, r \in \mathcal{R}\ ,\ \exists\, \pi_r^* \in \mathcal{F}(r, \mathcal{A})\ :\ \pi^* = \bigcup_{r \in \mathcal{R}} \chi_r \cdot \pi_r^*$$

Ainsi, toute politique optimale sur le MDP global correspond à un ensemble de politiques locales optimales au niveau des régions. Par conséquent, l'optimisation du

MDP abstrait produira une politique globale optimale à condition que l'ensemble de politiques locales données dans le modèle abstrait contient au moins une politique locale optimale par région au regard du MDP global.

Dans chaque région, il est intuitivement possible de générer une politique optimale dans le contexte global, en calculant la fonction de valeur du MDP dans la région par programmation dynamique, les fonctions de valeur des états périphériques étant initialisées à leur valeur optimale dans le MDP global. En effet, si les fonctions de valeur des états périphériques sont initialement optimales, la programmation dynamique restreinte à la région propage, dans cette région, les mêmes valeurs optimales que la programmation dynamique sur l'espace d'états total.

Ainsi, [23] propose de définir un MDP local dans chaque région, dont l'optimisation revient à propager la fonction de valeur des états périphériques, à l'intérieur de la région (cf. définition 5 et figure 2.4). Cette fonction de valeur est a priori inconnue sur les états périphériques, si bien qu'elle paramètre le modèle du MDP local, qualifié à ce titre de *générique*. Nous appellerons les fonctions de valeur des états périphériques *valeurs périphériques*.

Définition 5 MDP local générique d'une région

Soient un MDP $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, un partitionnement $\mathcal{R}$ de l'espace d'états $\mathcal{E}$ et une région $r \in \mathcal{R}$.

Le MDP local générique $\left\langle \overline{\mathcal{E}}, \overline{\mathcal{A}}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \right\rangle^{(\lambda_e)_{e \in Sper(r)}}$ de la région r est défini par :

- $\overline{\mathcal{E}} = r \cup Sper(r) \cup \{\alpha\}$ où α est un nouvel état absorbant ;
- $\overline{\mathcal{A}} = \mathcal{A}$;
- $\overline{\mathcal{T}} = \left(\overline{T}^a\right)_{a \in \mathcal{A}}$ tel que :

$$\forall\, a \in \mathcal{A}\,, \forall (e, e') \in \overline{\mathcal{E}}^2\,,$$
$$\overline{T}^a(e, e') = \chi_{(Sper(r) \cup \{\alpha\}) \times \{\alpha\}}(e, e') + \chi_{r \times (r \cup Sper(r))}(e, e') \cdot T^a(e, e')$$

- $\overline{\mathcal{R}} = \left(\overline{R}^a\right)_{a \in \mathcal{A}}$ tel que :

$$\forall\, a \in \mathcal{A}\,, \forall (e, e') \in \overline{\mathcal{E}}^2\,,$$
$$\overline{R}^a(e, e') = \chi_{Sper(r) \times \{\alpha\}}(e, e') \cdot \lambda_e + \chi_{r \times (r \cup Sper(r))}(e, e') \cdot R^a(e, e')$$

Il est prouvé [23] (cf. théorème 5) que la politique locale solution du MDP local d'une région donnée, dont les valeurs périphériques sont égales à la fonction de valeur du MDP global, est la restriction de la politique optimale globale à cette région. Ainsi, calculer un ensemble suffisant de politiques locales, au sens où au moins une politique locale de chaque région est optimale au regard du MDP global, revient à

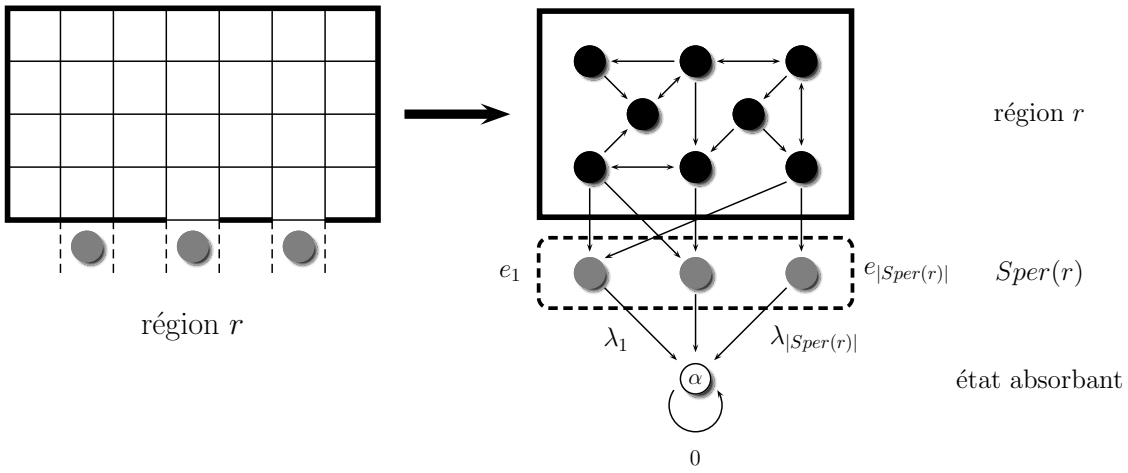


FIG. 2.4 – MDP local générique d'une région r, paramétré par les fonctions de valeur inconnues $(\lambda_e)_{e \in Sper(r)}$ des états périphériques de la région.

générer un ensemble suffisant de jeux de valeurs périphériques tel qu'au moins un jeu de valeurs périphériques de chaque région est égal à la fonction de valeur globale sur les états périphériques de la région.

Théorème 5 *Politique locale optimale [23]*

Soient un MDP $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, un partitionnement $\mathcal{R}$ de l'espace d'états $\mathcal{E}$, et V_γ^* la fonction de valeur optimale du MDP.
Soit le MDP local $\left\langle \overline{\mathcal{E}}, \overline{\mathcal{A}}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \right\rangle^{(\lambda_e)_{e \in Sper(r)}}$ tel que : $\left\| \lambda - V^*_{\gamma\,|_{Sper(r)}} \right\|_\infty \leqslant \epsilon$.
La fonction de valeur $V_\gamma^{\pi^*}$ de la politique locale π^* solution du MDP local vérifie :

$$\left\| V_\gamma^{\pi^*} - V^*_{\gamma\,|_r} \right\|_\infty \leqslant \frac{2\epsilon\gamma}{1-\gamma}$$

L'ensemble de recherche des jeux de valeurs périphériques est continu donc de dimension infinie. Néanmoins, il est borné par les valeurs minimale et maximale de la fonction de valeur sur l'ensemble du MDP [23] :

$$\underbrace{\frac{\min\limits_{e \in \mathcal{E}} \max\limits_{a \in \mathcal{A}} \sum\limits_{e' \in \mathcal{E}} T^a(e,e') \cdot R^a(e,e')}{1-\gamma}}_{\lambda_{min}} \leqslant V_\gamma^* \leqslant \underbrace{\frac{\max\limits_{e \in \mathcal{E}} \max\limits_{a \in \mathcal{A}} \sum\limits_{e' \in \mathcal{E}} T^a(e,e') \cdot R^a(e,e')}{1-\gamma}}_{\lambda_{max}}$$

Algorithmes de génération des valeurs périphériques

La fonction de valeur étant une fonction continue, l'ensemble des jeux de valeurs périphériques possibles est de dimension infinie. Pourtant, le nombre maximum de politiques locales de chaque région r est fini, égal à $|\mathcal{A}|^{|r|}$. Par conséquent, plusieurs jeux différents de valeurs périphériques produisent la même politique locale lors de l'optimisation du MDP local.

Ainsi, connaissant les bornes λ_{min} et λ_{max} des valeurs périphériques, deux méthodes de génération des jeux de valeurs périphériques ont été proposées :

- *recherche dans l'espace des valeurs périphériques* [23] : l'espace des valeurs périphériques est discrétisé, puis des politiques locales, chacune solution du MDP local paramétré par un point de la discrétisation, sont générées ; les politiques locales redondantes sont éliminées
- *recherche dans l'espace des politiques locales* [24] : un cache de politiques locales est construit de manière incrémentale en rajoutant au cache une nouvelle politique locale à chaque fois qu'il existe un point de l'espace des valeurs périphériques pour lequel aucune politique locale du cache n'est solution du MDP local paramétré par ce point.

La recherche dans l'espace des valeurs périphériques nécessite une discrétisation de cet espace entre les bornes λ_{min} et λ_{max} . Cette méthode présente deux inconvénients majeurs, qui la rendent inapplicables lorsque les régions sont fortement communiquantes :

- Aucune des politiques locales n'est garantie d'être optimale au regard du MDP global, car la politique locale optimale peut correspondre à un point, dans l'espace des valeurs périphériques, compris entre deux points de discrétisation. Pour une optimalité à ϵ près, d'après le théorème 5, la discrétisation doit avoir un pas d'au plus $\frac{\epsilon(1-\gamma)}{\gamma}$.
- Si une région a q sorties possibles, c'est-à-dire que l'espace des valeurs périphériques est $[\lambda_{min}; \lambda_{max}]^q$, l'optimalité à ϵ près nécessite de générer au moins $\left((\lambda_{max} - \lambda_{min})\, E_{\mathbb{N}}\left(\frac{\gamma}{\epsilon(1-\gamma)}\right)\right)^q$ politiques locales, ce qui est non seulement exponentiel en le nombre de sorties, mais qui tend de plus vers l'infini quand la précision augmente ($\epsilon \to 0$ et $\gamma \to 1$).

Aussi, il a été démontré que la recherche dans l'espace des politiques locales est bien plus intéressante et performante [24, 57, 21]. Elle est basée sur une approche par programmation linéaire, où les valeurs périphériques sont dans les contraintes d'un problème d'optimisation linéaire. Or, bien que la complexité des algorithmes de programmation linéaire, tel le `SIMPLEX`, soit théoriquement exponentielle en le nombre de contraintes dans le pire cas, il a été prouvé récemment qu'elle est *en moyenne polynomiale* en le nombre de contraintes, et que le pire cas est extrêmement rare [58]. Ainsi, dans la majorité des situations, la complexité de l'algorithme

de recherche dans l'espace des politiques locales [24], que nous présentons dans la suite, est polynomiale en le nombre de contraintes alors que celle de l'algorithme de recherche dans l'espace des valeurs périphériques [23] est exponentielle.

La recherche dans l'espace des politiques locales repose sur la notion de *politique locale dominante* (cf. definition 6 et figure 2.5) d'une région, qui maximise la fonction de valeur de chaque état interne de la région, sur l'ensemble des états et sur l'espace continu des valeurs périphériques (pour un jeu de valeurs périphériques λ donné, le maximum sur les actions de la fonction de valeur est obtenu pour la politique locale qui optimise le MDP local générique instancié par λ). En particulier, la politique locale optimale π^* au regard du MDP global correspond à la politique locale dominante $\overline{\pi}^*$ pour le jeu de valeurs périphériques $\left(V^*_\gamma(e)\right)_{e \in Sper(r)}$:

$$\forall\, e \in r\ ,\ \pi^*(e) = \overline{\pi}^* \left(e, V^*_{\gamma\,|_{Sper(r)}}\right)$$

Définition 6 Politique locale dominante

Soient un MDP $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, un partitionnement $\mathcal{R}$ de l'espace d'états $\mathcal{E}$, et une région $r \in \mathcal{R}$.
Soit $\left\langle \overline{\mathcal{E}}, \overline{\mathcal{A}}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \right\rangle^{(\lambda_e)_{e \in Sper(r)}}$ le MDP local générique de la région r.
Soient $(\lambda_{min}, \lambda_{max})$ les bornes de la fonction de valeur du MDP global sur l'espace d'états total.
La politique locale dominante $\overline{\pi}^*_r$ de la région r est la fonction de $r \times [\lambda_{min}; \lambda_{max}]^{Sper(r)}$ dans $\mathcal{A}$ définie par :

$$\forall\, e \in r\ ,\ \forall\, \lambda \in [\lambda_{min}; \lambda_{max}]^{Sper(r)}\ ,\ \overline{\pi}^*_r(e, \lambda) = \pi^*_{\left\{\left\langle \overline{\mathcal{E}}, \overline{\mathcal{A}}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \right\rangle^{\lambda}\right\}}(e)$$

où $\pi^*_{\left\{\left\langle \overline{\mathcal{E}}, \overline{\mathcal{A}}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \right\rangle^{\lambda}\right\}}$ est la politique locale optimisée du MDP local de la région r, dont le vecteur de valeurs périphériques est fixé à λ.

Or, comme le montre la figure 2.5, il est prouvé que la politique locale dominante d'une région, pour chaque état de la région, est une fonction croissante linéaire par morceaux des valeurs périphériques [24]. La démonstration de ce théorème repose sur la linéarité de la fonction de valeur d'une politique locale quelconque en fonction des valeurs périphériques.

Par conséquent, le nombre de politiques locales possibles d'une région r étant fini (majoré par $|\mathcal{A}|^{|r|}$) et l'espace de recherche des valeurs périphériques étant borné, la politique locale dominante est constituée d'un ensemble fini Π_r de politiques locales, qui dominent chacune exclusivement sur un polygone $\mathcal{P}_\pi$ de l'espace des valeurs

Théorème 6 *Croissance et linéarité par morceaux de la fonction de valeur de la politique locale dominante [24]*

Soient un MDP $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, un partitionnement $\mathcal{R}$ de l'espace d'états $\mathcal{E}$, et une région $r \in \mathcal{R}$.
Pour chaque état interne de la région, la fonction de valeur de toute politique locale π est une fonction croissante et linéaire des valeurs périphériques :

$$\forall\ \lambda \in \mathbb{R}^{|Sper(r)|}\ ,\ V_e^\pi(\lambda) = K_e^\pi + \lambda^\top a_e^\pi \quad avec \quad \begin{cases} K_e^\pi = V_e^\pi(0,\ldots,0) \\ a_{e,1}^\pi = V_e^\pi(1,0,\ldots,0) - K_e^\pi \\ \qquad\qquad \vdots \\ a_{e,|Sper(r)|}^\pi = V_e^\pi(0,\ldots,0,1) - K_e^\pi \end{cases}$$

Pour chaque état interne de la région, la fonction de valeur de la politique locale dominante est donc une fonction croissante et linéaire par morceaux des valeurs périphériques.

périphériques (cf. figure 2.5) :

$$\overline{\pi}^*(e) = \bigcup_{\pi \in \Pi_r} \chi_{\mathcal{P}_\pi}(e) \cdot \pi(e)$$

Ainsi, la politique locale optimale d'une région a la même propriété que la politique optimale des MDP Partiellement Observables (POMDP), les valeurs périphériques *continues* de la région étant le pendant des valeurs continues de l'espace de croyance [59]. L'ensemble Π_r des politiques locales qui constituent la politique locale dominante peut dont être calculé de manière similaire aux algorithmes d'optimisation des POMDP [24], et en particulier de l'algorithme `Witness` [60, 35].

Partant d'une politique locale initiale, optimisée pour des valeurs périphériques minimales ($\lambda = \{\lambda_{min}\}^{|Sper(r)|}$), de nouvelles politiques locales sont ajoutées à l'ensemble courant de politiques locales Π_r à chaque fois qu'il existe un point de l'espace des valeurs périphériques pour lequel aucune des politiques de Π_r n'est optimale. Lorsqu'un tel point n'existe plus, Π_r contient toutes les politiques locales qui constituent la politique locale dominante, et donc en particulier la politique locale optimale au regard du MDP global.

Comme la politique locale dominante est linéaire par morceaux, le point de l'espace des valeurs périphériques, où l'ensemble courant de politique locale Π_r n'est pas optimal, est solution du système linéaire suivant, pour tout état interne $e \in r$, tout état d'entrée $t \in Eper(r)$, toute action $a \in \mathcal{A}$ et toute politique locale $\pi \in \Pi_r$ de la politique locale dominante courante [24] :

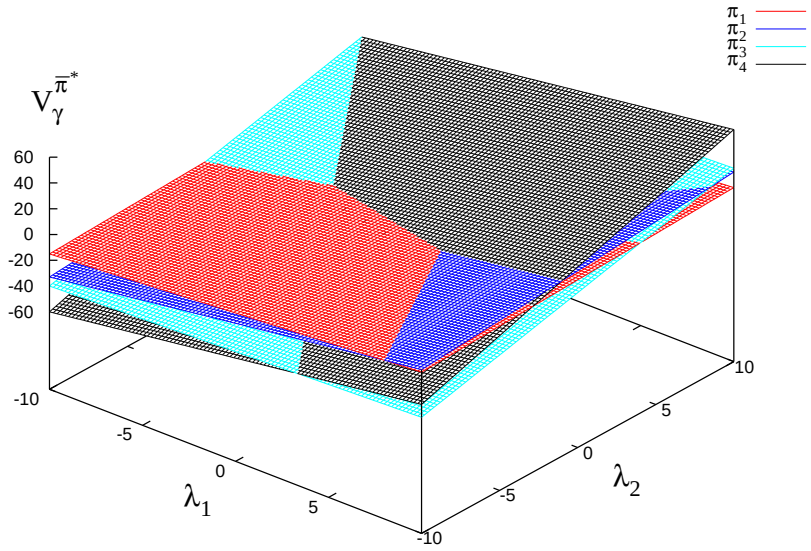


FIG. 2.5 – Fonction de valeur $V_\gamma^{\overline{\pi}^*}$ de la politique locale dominante $\overline{\pi}^*$, parmi 4 politiques locales, sur un état interne d'une région à 2 sorties

$$\begin{array}{ll} \texttt{Maximiser :} & \sum_{e' \in r} \overline{T}^a(e,e') \cdot \left(\overline{R}^a(e,e') + \gamma V_{e'}^{\pi}(\lambda)\right) - V_e^{\pi}(\lambda) \\ \texttt{Sujet à :} & \forall \pi' \in \Pi_r \ , \ V_t^{\pi'}(\lambda) \leqslant V_t^{\pi}(\lambda) \\ & \forall \ 1 \leqslant i \leqslant |Sper(r)| \ , \ \lambda_i \leqslant \lambda_{max} \end{array}$$

Autrement dit, pour chaque politique localement dominante (dans un polygone de l'espace des valeurs périphériques), le programme linéaire précédent permet de trouver le jeu de valeur périphérique dans le polygone où la politique domine, pour lequel l'erreur de Bellman est maximale (c'est-à-dire le point où cette politique localement dominante peut-être le plus amélioré). Ensuite, une nouvelle politique locale est ajouté au cache courant Π_r , optimisée pour le jeu de valeurs périphériques où l'erreur de Bellman est maximale parmi toutes les politiques localement dominantes.

Cette méthode est en pratique polynomiale en le nombre d'états périphériques des régions [24, 58, 21], si bien que nous l'utiliserons dans toute la suite, afin de réduire, dans les MDP factorisés, le sous-espace d'états énuméré engendré par certaines caractéristiques. Aussi, l'algorithme de décomposition de MDP énuméré que nous venons de décrire est détaillé plus loin (cf. algorithme 9 page 101).

Nous avons implémenté cet algorithme dans la librairie `graphMDP` (cf. annexe B page 275) à l'aide de la librairie d'optimisation linéaire GLPK [61], ainsi que l'algorithme de décomposition basé sur une recherche directe dans l'espace discrétisé des valeurs périphériques [23].

2.2 Factorisation d'un espace d'états défini par un ensemble de caractéristiques

2.2.1 Une description compacte et abstraite de l'espace d'états

Les techniques de décomposition de MDP sont efficaces lorsque l'espace d'états est structuré en régions faiblement communiquantes. Cependant, dans de nombreux problèmes de planification stochastique, l'espace d'états est si grand qu'il est impossible même de l'énumérer : le MDP, dans son formalisme classique *énuméré*, ne peut pas être construit. En effet, nous avons vu que le nombre d'états total est exponentiel en le nombre de caractéristiques de l'espace d'états (autonomie, position géographique, vitesse, vent, météo, etc...). Aussi, les méthodes qui visent à réduire l'espace d'états des MDP énumérés, comme la technique de décomposition que nous avons présenté, deviennent obsolètes.

Cependant, les caractéristiques de l'espace d'états suffisent à décrire tous les états, puisque chaque état est défini par une assignation de valeur, appelée *instanciation*, à toutes les caractéristiques (`autonomie`=totale, `position`=décollage, `vitesse`=0, `vent`=aucun, `météo`=dégagé, etc...). Ainsi, il est plus simple et nettement plus compact de représenter l'espace d'états par ses caractéristiques plutôt que par ses états énumérés. De plus, une instanciation partielle des caractéristiques définit un sous-ensemble d'états de l'espace total : par exemple, l'instanciation "`autonomie`=totale" correspond à tous les états où les autres caractéristiques (position, vitesse, vent, météo, etc...) prennent toutes leurs valeurs possibles. Plus un sous-espace d'états est grand, plus le nombre de caractéristiques instanciées pour le définir est petit.

Par conséquent, **une description de l'espace d'états par caractéristiques offre un cadre d'abstraction flexible et compact, qui permet de traiter les états par groupes de tailles variables, qui sont d'autant plus grands que le nombre de caractéristiques nécessaires pour les définir est faible** (cf. figure 2.6). Ceci est particulièrement utile pour les MDP, car la fonction de valeur et la politique sont souvent localement constantes sur des groupes d'états de grandes tailles. Autrement dit, sur chacun de ces groupes d'états, la fonction de valeur et la politique sont représentées par des instanciations partielles d'un nombre faible de caractéristiques, ce qui est particulièrement compact.

Dans cette section, nous présentons un modèle de MDP structuré [26, 31], où l'espace d'états est défini par un ensemble de caractéristiques appelées *variables d'état*, car l'instanciation de l'espace d'états peut être considéré comme une fonction des caractéristiques du modèle. L'espace d'états est dit *factorisé* par variables d'états, et le MDP structuré ainsi défini est nommé *MDP factorisé*. Les transitions ne sont plus des fonctions de l'espace d'états explicité dans lui-même, mais des fonctions

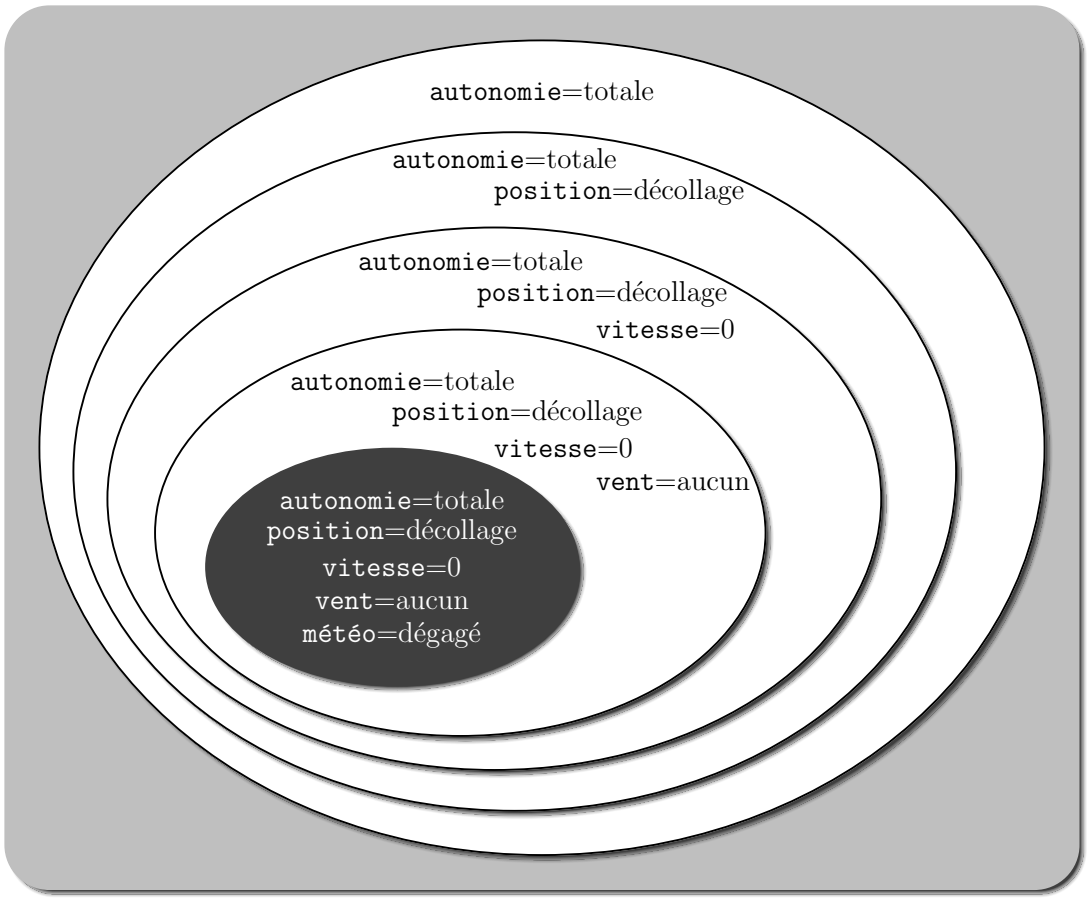


FIG. 2.6 – La taille d'un sous-espace d'états varie à l'opposé du nombre de caractéristiques instanciées nécessaires pour le décrire. L'ellipse grisée correspond à un état totalement instancié.

des variables d'état. Au besoin, la probabilité de passer d'un état instancié e à un état instancié e' peut être obtenue en calculant la probabilité jointe de passer de l'ensemble de variables d'état instanciées définissant e à celui définissant e'. Ainsi, les transitions sont définies par des Réseaux Bayésiens Dynamiques [62] (DBN), qui représentent pour chaque action l'influence stochastique des variables d'état avant la réalisation de l'action sur les variables d'état après sa réalisation.

D'autres modèles de MDP structurés existent, qui reposent sur des représentations différentes des caractéristiques de l'espace d'états. Certaines étendent des techniques de minimisation d'automates à l'abstraction de l'espace d'états des MDP [63, 64]. D'autres utilisent des systèmes de classeurs afin d'optimiser une stratégie dans un espace d'états où les caractéristiques sont représentées par des conditions propositionnelles [65, 66]. L'approche présentée dans cette section, que nous exploi-

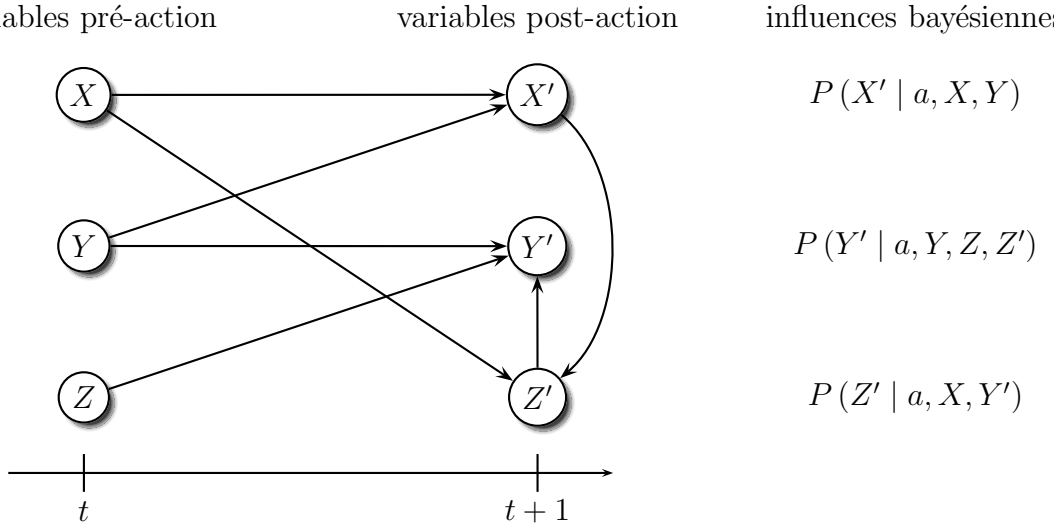


FIG. 2.7 – Exemple d'un Réseau Bayésien Dynamique d'un MDP factorisé en 3 variables d'état, pour une action a donnée

terons dans toute la suite, est connexe : elle est inspirée par des techniques de planification déterministe à l'aide du formalisme `STRIPS`, où l'espace d'états est défini par des propositions logiques [26, 67].

2.2.2 MDP factorisés par des variables d'état

Comme l'indique la définition 7, un MDP factorisé est défini sur un espace d'états factorisé en variables d'état de deux types, *pré-action* ou *post-action*, suivant que les variables d'état sont considérées respectivement avant ou après la réalisation d'une action. Pour chaque action, une variable d'état *post-action* dépend potentiellement de toutes les variables pré-action (dépendance *asynchrone*), ainsi que de toutes les variables post-action (dépendance *synchrone*).

Les dépendances stochastiques entre les variables d'état sont représentées par des Réseaux Bayésiens Dynamiques (DBN) [62], qui représentent l'influence bayésienne des variables d'état entre elles. Les nœuds de chaque DBN sont les variables d'état (cf. figure 2.7). Le processus stochastique étant supposé markovien, chaque DBN ne contient que deux niveaux de nœuds : les nœuds correspondant aux variables pré-action, et ceux correspondant aux variables post-action. Les arêtes entre les nœuds indiquent une dépendance stochastique entre deux variables d'état. Elles sont orientées afin de préciser quelle variable d'état dépend de l'autre.

Pour une action a et une variable d'état post-action X'_i données, la fonction T_i^a représente la distribution de probabilité sur les valeurs de X'_i, c'est-à-dire l'influence

Définition 7 Processus Décisionnels de Markov Factorisé

Un Processus Décisionnel de Markov Factorisé est un quadruplet $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$:

- $\mathcal{V} = \bigotimes_{i=1}^{n} V_i$ est un produit cartésien de n ensembles de variables d'état stochastiques $(\{X_i, X'_i\})_{1 \leqslant i \leqslant n}$, qui contiennent chacun une variable d'état pré-action X_i et une variable d'état post-action X'_i ;
- $\mathcal{A}$ est l'ensemble des actions de l'agent, chaque action étant un Réseau Bayésien Dynamique (DBN) ;
- $\mathcal{T} = \left(T_i^a\right)_{\substack{a \in \mathcal{A} \\ 1 \leqslant i \leqslant n}}$ est l'ensemble, pour chaque action (DBN), des influences stochastiques des variables d'état pré-action et post-action sur chaque variable d'état post-action :

$$\forall\, a \in \mathcal{A}\ ,\ \forall\, 1 \leqslant i \leqslant n\ ,\ T_i^a\ :\ \mathcal{V} \times (\mathcal{V} \setminus V_i) \to \mathcal{F}(V_i, [0;1]) \qquad \text{telle que :}$$

$$\forall\, (v, v') \in \mathcal{V} \times (\mathcal{V} \setminus V_i)\ ,\ T_i^a \left(\bigwedge_{j=1}^{n} v_j, \bigwedge_{j \neq i} v'_j \right)\ :$$

$$\begin{array}{rcl} V_i & \to & [0;1] \\ v'_i & \mapsto & P\left(X'_i = v'_i \mid a, \bigwedge_{j=1}^{n} (X_j = v_j), \bigwedge_{j \neq i} (X'_j = v'_j) \right) \end{array}$$

- $\mathcal{R} = \left(R^a\right)_{a \in \mathcal{A}}$ est l'ensemble, pour chaque DBN, des récompenses associées aux transitions des variables d'état pré-action vers les variables d'état post-action :

$$\forall\, a \in \mathcal{A}\ ,\ R^a\ :\ \begin{array}{rcl} \mathcal{V}^2 & \to & \mathbb{R} \\ (v, v') & \mapsto & r\left(\bigwedge_{i=1}^{n} (X'_i = v'_i) \mid a, \bigwedge_{i=1}^{n} (X_i = v_i) \right) \end{array}$$

bayésienne des variables d'état pré-action $(X_j)_{1 \leqslant j \leqslant n}$ et des autres variables post-action $\left(X'_j\right)_{j \neq i}$ sur X'_i. Dans les MDP énumérés (cf. définition 2 page 17), une seule fonction de transitions suffit par action pour décrire la dynamique du système. Au contraire, dans les MDP factorisés, la dynamique du système est détaillée au niveau des caractéristiques de l'espace d'états, si bien que chaque DBN définit autant de fonctions de transitions que de variables d'état post-action.

Chaque fonction de transition T_i^a est une distribution de probabilité sur les valeurs de la variable d'état X'_i , elle-même fonction des variables d'état dont dépend X'_i. Par commodité, T_i^a est définie sur toutes les variables pré-action et les variables post-action différentes de X'_i, bien que X'_i ne dépende en général pas de toutes les variables d'état.

Les fonctions de transition peuvent être définies sous forme d'*arbres de décision*, qui représentent des fonctions de variables d'état (cf. définition 8). Puisque chaque fonction de transition T_i^a est à valeur dans l'ensemble des fonctions de V_i dans $[0;1]$, les feuilles de l'arbre de décision qui définit T_i^a sont éléments de $\mathcal{F}(V_i, [0;1])$. Comme le montre la figure 2.8, les feuilles de cet arbre sont des listes qui représentent la distribution de probabilité sur les valeurs de la variable X'_i, connaissant les valeurs instanciées des variables qui sont dans les nœuds parents.

Définition 8 Arbre de décision

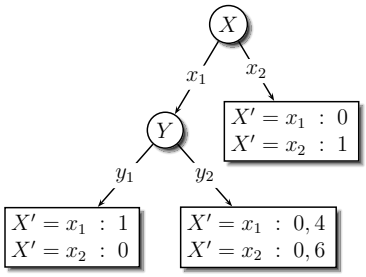
Soit $\mathcal{V} = (X_i)_{1 \leqslant i \leqslant n}$ un ensemble de n variables d'état, $\mathcal{H}$ un ensemble quelconque et une fonction $f : \mathcal{V} \to \mathcal{H}$.
L'arbre A_f de décision de la fonction f est un arbre dont les nœuds sont éléments de $\mathcal{V}$ et les feuilles éléments de $\mathcal{H}$, défini récursivement par :

- $\forall\ 1 \leqslant i < n$:
  - $A_f\left(X_1 = v_1, \ldots, X_i = v_i\right).racine() = X_{i+1}$
  - $\forall\ v_{i+1} \in V_{i+1}\ ,\ A_f\left(X_1 = v_1, \ldots, X_i = v_i\right).racine().fils(v_{i+1}) = A_f\left(X_1 = v_1, \ldots, X_{i+1} = v_{i+1}\right)$
- $A_f\left(X_1 = v_1, \ldots X_n = v_n\right) = f\left(X_1 = v_1, \ldots X_n = v_n\right)$

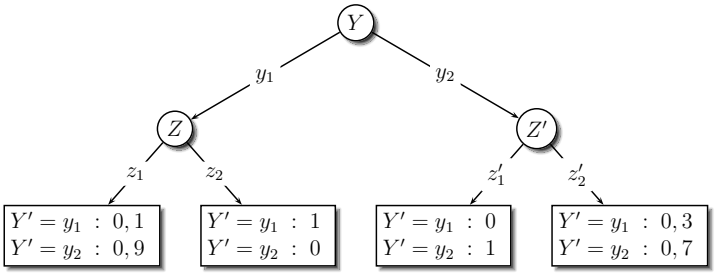
L'avantage des arbres de décision réside dans la compacité de la représentation : chaque feuille correspond à un sous-ensemble d'états défini par l'instanciation des variables des nœuds parents. Ce sous-ensemble d'états est d'autant plus grand que le nombre de nœuds parents est faible. La représentation sous forme d'arbres de décision est donc particulièrement intéressante si les fonctions ne dépendent pas de toutes les variables d'état. Par exemple, la fonction de transitions T_X^a de la figure 2.8 ne dépend que de 2 variables parmi 5 possibles, si bien que l'arbre de décision correspondant est particulièrement compact (4 feuilles au maximum, contre 32 si la fonction de transition dépend des 5 variables possibles). Autrement dit, les arbres de décision sont d'autant plus compacts que les DBN sont creux, puisque les arêtes des DBN représentent les dépendances entre les variables d'état.

La fonction de récompense peut être également définie sous forme d'arbre de décision, puisqu'elle représente une fonction des variables pré-action et post-action à valeurs réelles (cf. figure 2.8). De même, les fonctions de valeur et les politiques du MDP sont a priori des fonctions des variables d'état, si bien qu'elles seront définies elles aussi par des arbres de décision. Leurs arbres sont en général compacts, car la fonction de valeur est souvent localement constante sur des sous-espaces d'états assez grands [26, 31]. Le but des algorithmes d'optimisation des MDP factorisés est d'instancier le moins de variables d'état possibles dans la construction des arbres de la fonction de valeur et de la politique, au cours de l'optimisation du MDP.

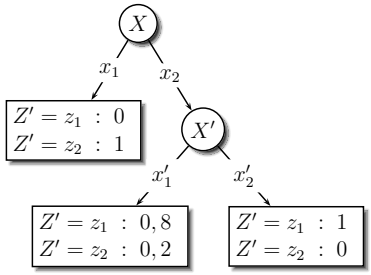
Arbre de transition de T_X^a



Arbre de transition de T_Y^a



Arbre de transition de T_Z^a



Arbre de récompense de R^a

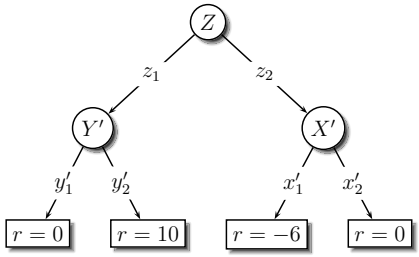


FIG. 2.8 – Exemple d'arbres de transition qui définissent les fonctions de transitions du DBN représenté dans la figure 2.7, et d'arbre de récompense

2.2.3 Algorithmes structurés pour l'optimisation des MDP factorisés

Algorithme `SPI`

Les premiers algorithmes qui ont été proposés pour optimiser des MDP factorisés sont des variantes de l'algorithme `SPI` («Structured Policy Iteration») [26, 27, 29, 30], qui définit et utilise des opérations algébriques sur les arbres de décision. Toute opération algébrique $T_1 * T_2$ entre deux arbres T_1 et T_2 s'effectue en trois temps :

1. concaténation de T_2 à chaque feuille f_1 de T_1 ;
2. f_1 est transmis aux feuilles de chaque sous-arbre T_2 concaténé, et le résultat de l'opération $f_1 * f_2$ remplace les anciennes feuilles ;
3. l'arbre obtenu est simplifié en enlevant les nœuds redondants (nœuds identiques dans un même chemin de la racine vers une feuille) et les sous-arbres égaux.

La programmation dynamique est réalisée par un procédé appelé *régression décisionnelle*. A chaque pas de temps t du processus d'optimisation, l'arbre de valeur courant V_t^* est analysé afin de connaître les variables d'état dont il dépend. Lors de la mise à jour de la fonction de valeur au pas de temps $t-1$, ces variables sont considérées comme des variables post-action, puisqu'elles appartiennent à un arbre défini au pas de temps suivant t.

Aussi, pour chaque action a, les arbres de transition de ces variables sont multipliés entre eux afin de calculer à la fois les sous-ensembles d'état permettant d'arriver dans les sous-ensembles d'états représentés par les feuilles de V_t^*, et les probabilités correspondantes d'y arriver. L'arbre de probabilités jointes obtenu est, suivant l'équation de Bellman (cf. théorème 3 page 22), multiplié par V_t^* puis ajouté à l'arbre de récompense de l'action a, afin de calculer la fonction de valeur V_{t-1}^a de l'action a à l'instant $t-1$.

Néanmoins, la programmation dynamique par régression décisionnelle gère mal les dépendances synchrones (entre les variables post-action) [29]. Les arbres de transition doivent être préalablement analysés afin d'éliminer les dépendances synchrones à l'aide de la relation de Bayes :

$$P(A \mid C) = \sum_B P(A \mid B, C) \cdot P(B \mid C)$$

La difficulté vient de la nécessité de déterminer l'ordre d'élimination des variables post-action, car l'élimination d'une variable post-action peut inclure dans l'arbre une nouvelle dépendance synchrone [29].

De plus, des algorithmes récents basés sur les diagrammes algébriques de décision (ADD), que nous détaillons dans le paragraphe suivant, ont montré leur supériorité sur les algorithmes de type `SPI` [31]. Ainsi, nous n'utiliserons ni les arbres de décision

ni l'algorithme `SPI` dans la suite, pour l'optimisation des MDP factorisés. Nous supposerons que les arbres de décision sont donnés dans le modèle, mais qu'ils sont transformés en ADD par un procédé que nous verrons plus loin.

Diagrammes de Décision

Les diagrammes de décision [68, 69] sont des structures algébriques compactes qui représentent des fonctions de variables binaires de deux types possibles :

- Diagrammes de Décision Binaires (BDD) : fonctions de variables binaires à valeur binaire ($\mathcal{F}(\{0,1\}^n, \{0,1\})$, $n \in \mathbb{N}^*$) ;
- Diagrammes de Décision Algébriques (ADD) : fonctions de variables binaires à valeur réelle ($\mathcal{F}(\{0,1\}^n, \mathbb{R})$, $n \in \mathbb{N}^*$).

Comme pour les arbres de décision, les diagrammes de décision sont des graphes dont les nœuds sont les variables des fonctions qu'ils représentent, et dont les feuilles sont les valeurs de ces fonctions pour l'instanciation des variables parentes. Les principales différences avec les arbres de décision sont :

- les variables sont *uniquement* binaires, car tous les calculs sont optimisés pour des opérations logiques ;
- les variables sont ordonnées, afin de faciliter la concaténation et la simplification de diagrammes de décision ;
- tous les sous-diagrammes ne sont enregistrés qu'une seule fois en mémoire : deux sous-diagrammes égaux ne font qu'un (héritage multiple dans le graphe des nœuds parents de ces deux sous-diagrammes) ;
- une «valeur de fond» préalablement définie n'est jamais enregistrée en mémoire (par défaut : 0)

A titre d'exemple, le diagramme de décision R^a de la figure 2.10 (en haut à droite), est équivalent à l'arbre de récompense de la figure 2.8. La feuille grisée correspond à la valeur de fond. Dans le diagramme de décision en bas à gauche de la figure 2.10, le quatrième nœuds Z en partant de la gauche a deux parents différents (les deux parents ont en aucun un même sous-diagramme dont la racine est ce nœud Z).

L'héritage multiple renforce la compacité des diagrammes. Nous avons vu que les arbres de décision sont une représentation compacte des fonctions de variables d'état, car chaque feuille correspond à un sous-ensemble d'états plus ou moins grand, suivant le nombre de nœuds parents de la feuille. En plus de cette propriété, les diagrammes de décision diminuent encore le nombre de feuilles à énumérer grâce à la fusion de tous les sous-arbres égaux.

Par ailleurs, la valeur de fond est une notion commune avec les matrices creuses [47], qui sont des matrices où la valeur 0 n'est jamais enregistrée. Or, l'utilisation des matrices creuses dans les MDP énumérés est, la plupart du temps, plus efficace que l'utilisation des matrices pleines, car la densité en éléments nuls dans les matrices de transitions stochastiques est très faible [21]. Aussi, cette idée a été exploitée dans

les MDP factorisés basés sur les ADD [31], afin de ne générer qu'une seule fonction de transition complète par action, qui représente la matrice de transition creuse du MDP énuméré correspondant. Le diagramme de décision de cette fonction de transition complète est appelé *Diagramme D'action Complet* (CAD) [31]. Bien entendu, cette approche est rédhibitoire avec les arbres de décision, car l'arbre de transition complet correspondrait à une matrice pleine, dont l'énumération est impossible pour des problèmes de grande taille.

Pour une action a donnée, le CAD est obtenu en multipliant entre eux les ADD équivalents aux arbres de transition des variables d'état post-action (cf. figure 2.9). L'arbre de transition d'une variable d'état X_i donnée représente une fonction à valeur dans $\mathcal{F}(V_i, [0;1])$, alors que les ADD encodent des fonctions à valeur dans $\mathbb{R}$. Ainsi, la construction de l'ADD équivalent nécessite deux étapes (cf. figure 2.9) :

1. Construction d'un ADD pour chaque valeur de la variable post-action, en connaissant dans chaque feuille les valeurs instanciées des autres variables d'état ; en supposant cette variable binaire :
   - $T^a_{X_i}(X'_i)$: $\begin{array}{lcl} \mathcal{V} \times (\mathcal{V} \setminus V_i) & \rightarrow & [0;1] \\ (v, v') & \mapsto & P\left(X'_i = \textbf{true} \mid a, \bigwedge_{j=1}^{n} (X_j = v_j), \bigwedge_{j \neq i} (X'_j = v'_j)\right) \end{array}$
   - $T^a_{X_i}(\overline{X'_i})$: $\begin{array}{lcl} \mathcal{V} \times (\mathcal{V} \setminus V_i) & \rightarrow & [0;1] \\ (v, v') & \mapsto & P\left(X'_i = \textbf{false} \mid a, \bigwedge_{j=1}^{n} (X_j = v_j), \bigwedge_{j \neq i} (X'_j = v'_j)\right) \end{array}$

   Ces deux ADD ont la même structure (celle de l'arbre de décision initial), mais ils diffèrent par leurs feuilles, qui sont deux à deux complémentaires par rapport à 1.
2. Calcul de l'ADD représentant la distribution de probabilités sur les valeurs de X' *a posteriori*, en connaissant dans chaque feuille les valeurs instanciées des autres variables d'état :

$$T^a_{X_i} : \begin{array}{lcl} \mathcal{V}^2 & \rightarrow & [0;1] \\ (v, v') & \mapsto & P\left(X'_i = v'_i \mid a, \bigwedge_{j=1}^{n} (X_j = v_j), \bigwedge_{j \neq i} (X'_j = v'_j)\right) \end{array}$$

   La formule logique permettant de calculer cette fonction est :

$$T^a_{X_i} = \left(\chi_{X'_i} \cdot T^a_{X_i}(X'_i)\right) + \left(\chi_{\overline{X'_i}} \cdot T^a_{X_i}(\overline{X'_i})\right)$$

   où χ_E est la fonction indicatrice d'un sous-ensemble d'états :

$$\forall\, e \in \mathcal{V}\ ,\ \chi_E(e) = \begin{cases} 1 \text{ si } e \in E \\ 0 \text{ sinon} \end{cases}$$

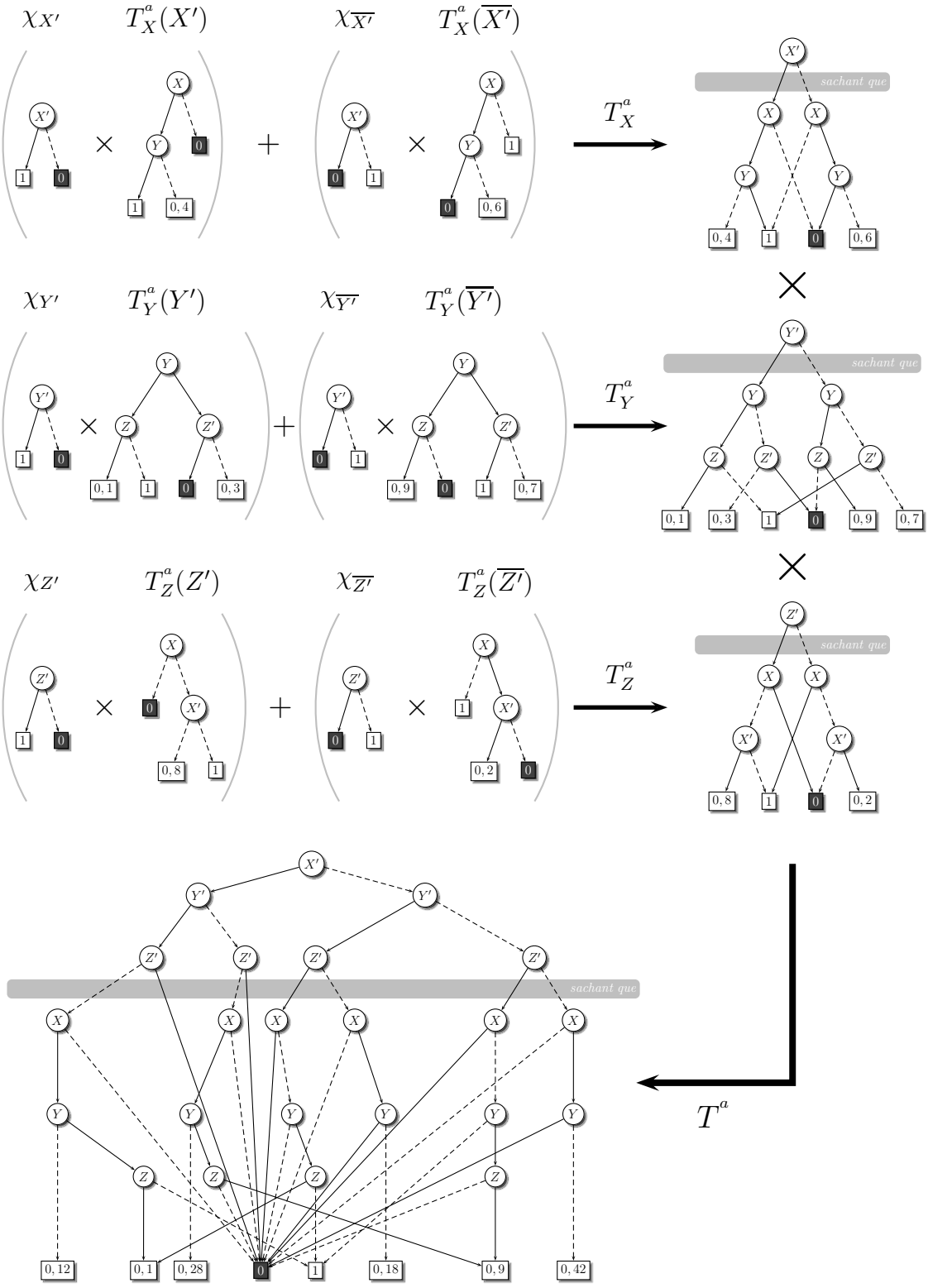


FIG. 2.9 – Exemple de construction du Diagramme d'Action Complet (CAD), pour l'action dont les arbres de transition sont représentés dans la figure 2.8. Le CAD est le produit des ADD correspondant aux arbres de transition de chaque variable d'état, chacun de ces ADD étant la somme sur les valeurs **true** et **false** de l'arbre de transition correspondant.

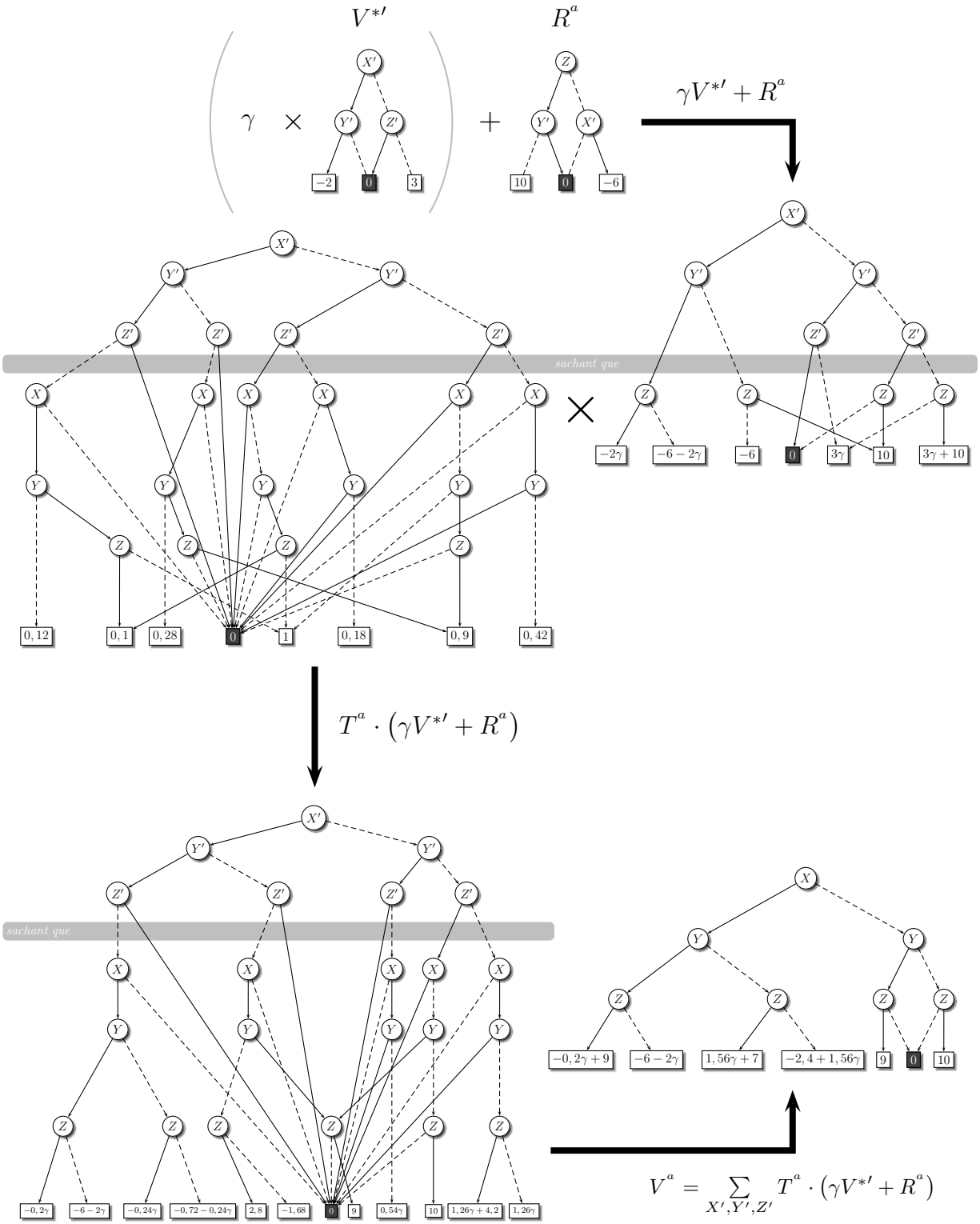


FIG. 2.10 – Exemple de construction de la fonction de valeur d'une action, dont le diagramme d'action complet a été calculé dans la figure 2.9, et dont l'arbre de récompense est représenté dans la figure 2.8

La fonction $T^a_{X_i}$, suivant la formule logique précédente, peut être calculée très simplement à l'aide d'une librairie informatique de diagramme de décision, telle CUDD [70]. Quelque soit la librairie considérée, les opérations sur les BDD et les ADD reposent sur des concaténations et des simplifications récursives de diagrammes, comme pour les arbres de décision [68, 69, 70]. Les paramètres des routines sont donc des fonctions, ce qui explique l'écriture de $T^a_{X_i}$ sous forme d'opérations algébriques sur des fonctions. Dans toute les suite, nous utiliserons les notations de la librairie CUDD, très largement répandue, mais celles-ci devraient correspondre également aux autres librairies de diagrammes de décision.

Les routines nécessaires pour le calcul de $T^a_{X_i}$ sont (cf. algorithme 3) :

- `addIthVar`$(2i)$: renvoie l'ADD $\chi_{X'_i}$ (l'indice est doublé afin de référencer une variable post-action) ;
- `addCmpl`$(\chi_{X'_i})$: renvoie l'ADD $\chi_{\overline{X'_i}}$, complémentaire de l'ADD $\chi_{X'_i}$;
- `addApply`$(\texttt{addPlus}, d_1, d_2)$: renvoie la somme de deux ADD d_1 et d_2 (`addApply` est la routine générique de récursivité, dont le premier paramètre est l'opérateur algébrique qui doit être appliqué sur les feuilles) ;
- `addApply`$(\texttt{addTimes}, d_1, d_2)$: renvoie le produit de deux ADD d_1 et d_2 .

Les opérations sur les BDD et les ADD sont dites *symboliques*, car les paramètres des routines sont des fonctions symboliques non instanciées. Autrement dit, les états sont traités par lots.

Le diagramme d'action complet T^a peut alors être calculé, comme produit de toutes les distributions de probabilités a posteriori des variables post-action (cf. figure 2.9 et algorithme 3) :

$$\forall\,(v,v')\in\mathcal{V}^2\ ,\ P\left(\bigwedge_{i=1}^{n}(X'_i=v'_i)\mid a,\bigwedge_{i=1}^{n}(X_i=v_i)\right)=$$

$$\prod_{i=1}^{n}P\left(X'_i=v'_i\mid a,\bigwedge_{j=1}^{n}(X_j=v_j),\bigwedge_{j\neq i}(X'_j=v'_j)\right)$$

$$\Longleftrightarrow\qquad T^a=\prod_{i=1}^{n}T^a_{X_i}$$

Ce résultat se démontre en écrivant le produit des distributions de probabilités a posteriori dans l'ordre des dépendances synchrones, et en supposant que les dépendances synchrones ne sont pas cycliques (sinon, le modèle est inconsistant).

Un tel formalisme, qui n'est pas réalisable — rappelons-le — avec les arbres de décision, est très intéressant car il permet d'optimiser les MDP factorisés en utilisant *directement* l'équation de Bellman (cf. théorème 3 page 22). Les opérations internes aux BDD et aux ADD sont transparentes, et gérées de manière optimale par la librairie de diagramme de décision.

Algorithme 3 : Fonction `CalculeDiagrammeActionComplet` [33]

Données : $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$
Résultat : $\left\{T^a\right\}_{a \in \mathcal{A}}$ (liste d'ADD)
début
  pour $1 \leqslant i \leqslant n$ **faire**
    $addX'_i \leftarrow$ `addIthVar`$(2i)$;
    $add\overline{X'_i} \leftarrow$ `addCmpl`$(addX'_i)$;
  pour $a \in \mathcal{A}$ **faire**
    $T^a \leftarrow$ `ReadOne`();
    pour $1 \leqslant i \leqslant n$ **faire**
      $add\_temp\_1 \leftarrow$ `addApply`$(\texttt{addTimes}, addX'_i, T^a_{X_i}(X'_i)$;
      $add\_temp\_2 \leftarrow$ `addApply`$(\texttt{addTimes}, add\overline{X'_i}, T^a_{X_i}(\overline{X_i}))$;
      $T^a_{X_i} \leftarrow$ `addApply`$(\texttt{addPlus}, add\_temp\_1, add\_temp\_2)$;
      $T^a \leftarrow$ `addApply`$(\texttt{addTimes}, T^a, T^a_{X_i})$;
  retourner $\left\{T^a\right\}_{a \in \mathcal{A}}$;
fin

De plus, les dépendances synchrones sont implicitement contenues dans la distribution de probabilités jointes, si bien qu'elles ne nécessitent pas de traitement spécifique durant l'optimisation du MDP (contrairement aux arbres de décision [29]).

Algorithme `SPUDD`

L'algorithme `SPUDD` («Stochastic Planning Using Decision Diagrams», cf. algorithme 4) est une version symbolique de l'algorithme d'itération de la valeur, qui utilise les diagrammes de décision [32]. Alors que dans l'algorithme «classique» la fonction de valeur est mise à jour sur chaque état individuel, elle est actualisée dans `SPUDD` symboliquement sur l'ensemble des états. Ainsi, l'équation de Bellman doit être considérée symboliquement :

$$V^*_\gamma = \max_{a \in \mathcal{A}} \left\{ \sum_{v' \in \mathcal{V}} \left[T^a \cdot \left(R^a + \gamma V^*_\gamma \right) \right] \left(\wedge_{1 \leqslant i \leqslant n} (X'_i = v'_i) \right) \right\}$$

La mise à jour de la fonction de valeur nécessite trois étapes de calculs symboliques, dont les deux premières sont schématisées dans la figure 2.10 :

1. Calcul de la fonction (ADD) $f = T^a \cdot \left(R^a + \gamma V^*_\gamma \right)$ en utilisant la fonction récursive `addApply` avec les opérateurs algébriques `addPlus` (somme) et `addTimes` (produit) ;

2. Somme sur toutes les variables d'état post-action (états successeurs) à l'aide de la fonction `addExistAbstract`$(f, \textit{cube\_variables\_primees})$, où l'ADD *cube_variables_primees* représente la fonction $\chi_{\wedge_{1\leqslant i\leqslant n}(X'_i=\mathbf{true})}$; cette fonction peut être obtenue avec la routine `addComputeCube`$(\wedge_{1\leqslant i\leqslant n}\{X'_i\})$;
3. Calcul de la fonction maximum entre l'ADD obtenu précédemment et la fonction de valeur courante, avec la routine récursive `addApply` combinée à l'opérateur algébrique `addMaximum`.

La fonction de valeur ne contient que des variables pré-action, qui doivent être transposées en variables post-action pour l'itération suivante. La transformation entre les variables pré-action et post-action (et vice-versa) s'effectue très simplement à l'aide de la routine `addPermute`$(V^*_\gamma, \textit{unprimed\_to\_primed})$, où le paramètre *unprimed_to_primed* est un tableau qui indique, pour chaque variable pré-action, l'indice de la variable post-action correspondante.

Enfin, la politique est encodée sous la forme d'un ensemble de BDD (diagrammes de décision *binaires*), qui représentent chacun le sous-ensemble d'états où doit s'appliquer chaque action :

$$\pi = \left\{\chi_{\pi^{-1}(a)}\right\}_{a\in\mathcal{A}}$$

Cette représentation est plus appropriée qu'un seul ADD dont les feuilles seraient les indices des actions à appliquer dans le sous-ensemble d'états correspondant à chaque feuille, car les ADD ne sont pas pratiques pour des accès individuels aux feuilles des diagrammes. Ils sont en effet conçus et optimisés pour des accès symboliques par lots d'états.

D'après le théorème 3 de caractérisation des politiques optimales (page 22), une action a s'applique sur l'ensemble des états où elle améliore la fonction de valeur. Or, la fonction de différence $(diff)$ entre le maximum de la fonction de valeur courante V^*_γ et la valeur V^a_γ de l'action a vaut 0 sur l'ensemble des états où a améliore V^*_γ, et 1 ailleurs. Ainsi, après avoir transformé l'ADD $diff$ en BDD *bdd_diff* (routine `addBddPattern`), l'action a s'applique sur le complémentaire du sous-ensemble d'états représenté par bdd_diff :

$$\pi^a = \overline{\left\{e\in\mathcal{V}\ :\ V^*_\gamma(e) - V^a_\gamma(e) = 0\right\}} = \mathcal{V}\sqcup\left\{e\in\mathcal{V}\ :\ V^*_\gamma(e) - V^a_\gamma(e) = 0\right\}$$

en notant $\sqcup$ la relation logique «OU EXCLUSIF» ($\mathcal{V}$ est l'espace d'états entier).
Le «OU EXCLUSIF» est obtenu en appelant la routine `bddXor`(b_1, b_2) , qui calcule $b_1 \sqcup b_2$. Enfin, le sous-ensemble d'états π^a doit être enlevé des BDD de la politique des actions précédentes, car ces actions ne s'appliquent plus sur π^a (car π^a améliore la fonction de valeur). Pour une action précédente a', la politique $\tilde{\pi}^{a'}$ mise à jour vérifie la formule logique :

$$\tilde{\pi}^{a'} = \pi^{a'} \setminus \pi^a = \left(\pi^{a'} \cap \pi^a\right) \sqcup \pi^{a'}$$

Algorithme 4 : Fonction `SPUDD` [32]

Données : $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, γ , ϵ
Résultat : π (liste de BDD), V (ADD)
début
  $\{T^a\}_{a \in \mathcal{A}} \leftarrow$ `CalculeDiagrammeActionComplet`$(\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle)$;
  $cube\_variables\_primees \leftarrow$ `addComputeCube` $(\wedge_{1 \leqslant i \leqslant n} \{X'_i\})$;
  $V \leftarrow$ `ReadZero`();
  répéter
    $saveV \leftarrow V$;
    $V' \leftarrow$ `addPermute`$(saveV, unprimed\_to\_primed)$;
    pour $a \in \mathcal{A}$ **faire**
      $V^a \leftarrow$ `addApply`$(\texttt{addTimes}, V', \texttt{addConst}(\gamma))$;
      $V^a \leftarrow$ `addApply`$(\texttt{addPlus}, V^a, R^a)$;
      $V^a \leftarrow$ `addApply`$(\texttt{addTimes}, V^a, T^a)$;
      $V^a \leftarrow$ `addExistAbstract`$(V^a, cube\_variables\_primees)$;
      si $a = \mathcal{A}.premier()$ **alors**
        $V \leftarrow V^a$;
      sinon
        $V \leftarrow$ `addApply`$(\texttt{addMaximum}, V, V^a)$;
  jusqu'à `EqualSupNorm`$(V, saveV, \epsilon) =$ **true**;
  $V' \leftarrow$ `addPermute`$(V, unprimed\_to\_primed)$;
  pour $a \in \mathcal{A}$ **faire**
    $V^a \leftarrow$ `addApply`$(\texttt{addTimes}, V', \texttt{addConst}(\gamma))$;
    $V^a \leftarrow$ `addApply`$(\texttt{addPlus}, V^a, R^a)$;
    $V^a \leftarrow$ `addApply`$(\texttt{addTimes}, V^a, T^a)$;
    $V^a \leftarrow$ `addExistAbstract`$(V^a, cube\_variables\_primees)$;
    si $a = \mathcal{A}.premier()$ **alors**
      $V \leftarrow V^a$;
      $\pi^a \leftarrow$ `ReadOne`();
    sinon
      $V \leftarrow$ `addApply`$(\texttt{addMaximum}, V, V^a)$;
      $diff \leftarrow$ `addApply`$(\texttt{addMinus}, V, V^a)$;
      $bdd\_diff \leftarrow$ `addBddPattern`$(diff)$;
      $\pi^a \leftarrow$ `bddXor`$(\texttt{ReadOne}(), bdd\_diff)$;
      pour $a' = \mathcal{A}.premier()$ **jusqu'à** $a' = \mathcal{A}.precedent(a)$ **faire**
        $\pi^{a'} \leftarrow$ `bddXor`$(\texttt{bddAnd}(\pi^a, \pi^{a'}), \pi^{a'})$;
  retourner (π, V);
fin

Cas des actions non applicables sur tout l'ensemble d'états

L'algorithme `SPUDD` suppose que les actions s'appliquent chacune sur tout l'ensemble d'états. Si ce n'est pas le cas, une première solution consiste à modifier le modèle de récompense afin que la valeur $-\infty$ soit attribuée aux états où une action ne s'applique pas. Néanmoins, ce procédé rajoute des feuilles de valeur $-\infty$ à l'ADD de la fonction de valeur, ce qui peut augmenter sensiblement la taille de cet ADD dans certains cas.

Une meilleure solution repose sur la notion de *masque d'action*. Pour chaque action a, un BDD M^a représente le sous-ensemble d'états $\mathcal{D}^a$ sur lequel l'action a s'applique : $M^a = \chi_{\mathcal{D}^a}$. Le BDD M^a permet de masquer la fonction de valeur courante V sur $\overline{\mathcal{D}^a}$ lors du calcul du maximum entre V et la valeur V^a de l'action.

Néanmoins, la valeur de fond des ADD nécessite de calculer ce maximum avec précaution. En effet, l'ADD V^a est nul en dehors des états où s'applique l'action a (0 est valeur de fond par défaut), si bien que le maximum entre V et V^a peut être biaisé si V est négative sur $\overline{\mathcal{D}^a}$. Ainsi, la fonction de valeur doit être masquée par M^a, afin de conserver les éventuelles valeurs négatives de V sur $\overline{\mathcal{D}^a}$:

$$maxV = \chi_{\overline{\mathcal{D}^a}} \cdot V + \chi_{\mathcal{D}^a} \cdot \max(V, V^a)$$

Par conséquent, la ligne suivante dans `SPUDD` (cf. algorithme 4) :

$$V \leftarrow \texttt{addApply}(\texttt{addMaximum}, V, V^a);$$

doit être remplacée par les instructions ci-dessous :

$$\begin{array}{l}
\overline{M^a} \leftarrow \texttt{bddXor}(\mathcal{V}, M^a); \\
\overline{masque}V \leftarrow \texttt{addApply}(\texttt{addTimes}, V, \texttt{BddToAdd}(\overline{M^a})); \\
V \leftarrow \texttt{addApply}(\texttt{addMaximum}, V, V^a); \\
masqueV \leftarrow \texttt{addApply}(\texttt{addTimes}, V, \texttt{BddToAdd}(M^a)); \\
V \leftarrow \texttt{addApply}(\texttt{addPlus}, \overline{masque}V, masqueV);
\end{array}$$

Cas des variables d'état non binaires

Toute variable d'état non binaire peut être transformée en $E_{\mathbb{N}}(\ln_2(q))(+1)$ variables binaires, où q est le nombre valeurs possibles de cette variable, et $E_{\mathbb{N}}$ est la fonction "partie entière". En effet, $\ln_2(q)$ est le nombre réel minimum de variables binaires, dont les instanciations simultanées sont au moins au nombre de q :

$$\underset{r \in \mathbb{N}}{\operatorname{argmin}} \{2^r \geqslant q\} = \begin{cases} \ln_2(q) & \text{si } \ln_2(q) \in \mathbb{N} \\ E_{\mathbb{N}}(\ln_2(q)) + 1 & \text{sinon} \end{cases}$$

Il a été démontré [31] que le coût de cette transformation est, dans la plupart des cas, négligeable devant le coût d'utilisation d'un algorithme d'optimisation de MDP factorisés qui gère directement des variables non binaires, tel `SPI`.

Bilan

Nous avons présenté un modèle de planification proactive sous incertitudes des actions, appelé *Processus Décisionnels de Markov*. Ce formalisme vise à optimiser une stratégie pour un agent, c'est-à-dire à calculer le comportement global et optimal de l'agent, qui maximise la moyenne de ses gains accumulées lors du processus. Néanmoins, le modèle «classique» de MDP, où l'espace d'états est explicité et non structuré, est peu efficace pour des problèmes de grande taille. En effet, il n'exploite pas les spécificités de certaines parties de l'espace d'états, et l'énumération des états est parfois rédhibitoire car ils sont trop nombreux.

Ainsi, des modèles récents et structurés de MDP ont été proposés, afin d'optimiser la fonction de valeur des MDP sur des sous-ensembles d'états de petites tailles. Lorsqu'il est possible d'expliciter l'espace d'états, et qu'il peut être partitionné en régions faiblement communiquantes (par exemple, vallées ou montagnes dans un espace d'états géographiques), des techniques de *décomposition* de MDP permettent d'optimiser la stratégie par *programmation dynamique asynchrone* sur chaque région du partitionnement. L'intérêt de cette décomposition réside dans l'optimisation de sous-stratégies sur des sous-espace d'états de petites tailles. Elle permet en outre de construire un modèle réduit et abstrait de MDP, dont les états sont les régions du partitionnement, et les actions sont les sous-stratégies calculées dans chaque région.

D'autre part, il arrive souvent que l'espace d'états de l'agent soit décrit par un ensemble de caractéristiques (`position`, `vitesse`, `autonomie`), dont une assignation simultanée de valeurs définit un état individuel (`position`=décollage, `vitesse`=nulle, `autonomie`=totale). La taille de l'espace d'états étant exponentielle en le nombre de ses caractéristiques, il est primordial de disposer d'un modèle de MDP permettant d'optimiser la stratégie sans expliciter les états individuels. Aussi, les MDP *factorisés* ont été récemment développé, où les caractéristiques sont des variables d'état qui ne sont instanciées qu'au besoin au cours de l'optimisation du processus. L'intérêt des algorithmes d'optimisation des MDP factorisés réside dans l'instanciation partielle des variables, qui permet de traiter symboliquement les états par groupes d'état.

Cependant, les approches décomposée et factorisée de MDP sont efficaces pour des modèles *a priori* incompatibles d'espace d'états. Le formalisme théorique de la décomposition de MDP nécessite que l'espace d'états soit énuméré et partitionné

en régions faiblement communiquantes, alors que la factorisation de MDP suppose que l'espace d'états est défini implicitement (ou *en intention*) par un ensemble de variables d'état. Néanmoins, dans de nombreux problèmes tels ceux de déplacement et de prise d'information conjugués, l'espace d'états est globalement factorisé par variables d'état, mais le sous-espace d'états engendré par une des variables (navigation par exemple) est structuré en régions distinctes et faiblement communiquantes.

Par conséquent, dans les deux parties suivantes, nous proposons une formalisme de *décomposition factorisée*, qui consiste à appliquer des techniques de décomposition sur des sous-MDP énumérés, définis sur les sous-espaces d'états engendrés par certaines variables. Notre approche décomposée et factorisée sera utilisée dans deux buts différents :

- réduire la taille de l'espace d'états factorisé total, en partitionnant le sous-espace engendré par une variable de grande arité, dans le but de **faciliter l'optimisation du MDP** ; cette contribution peut être considérée comme une généralisation des techniques de décomposition de MDP énumérés aux MDP factorisés.
- réduire le nombre de valeurs possibles d'une variable d'état de grande arité, afin de **simplifier la modélisation du MDP**

Le seul point commun entre ces deux contributions, certes proches, est le partitionnement de l'ensemble des valeurs possibles d'une variable d'état, par des techniques de décomposition de sous-MDP énumérés.

Par ailleurs, si les MDP factorisés sont très performants sur des espaces d'états définis par caractéristiques, ils ne retardent que l'effet du «*curse of dimensionality*» [71], qui stipule que l'espace d'états augmente exponentiellement avec le nombre de variables d'état. En effet, la structure de la fonction de valeur se complexifie généralement avec l'augmentation de la taille de l'espace d'états, si bien que le nombre de sous-espaces d'états de valeurs différentes devient très important. Ceci se traduit par une augmentation sensible de la taille des ADD et des BDD qui, comme nous le verrons plus loin, devient rédhibitoire pour des problèmes de grande taille. Aussi, dans une dernière partie, nous proposerons un algorithme symbolique qui permet d'**optimiser le MDP sur un sous-ensemble d'états, connaissant des états initiaux et buts**.

Deuxième partie

Réduction de MDP factorisés par des techniques de décomposition de sous-MDP énumérés

Motivation

Les modèles factorisés de MDP, qu'ils soient basés sur les arbres de décision ou sur les Diagrammes de Décision Algébriques (ADD), sont difficiles à résoudre lorsque certaines variables d'état ont un grand nombre de valeurs possibles.

Dans le cas des arbres de décision contenant ds variables d'état d'arité quelconque, les arbres obtenus sont très larges et donc très grands, ce qui rend l'optimisation du MDP particulièrement difficile : les opérations de concaténation et de simplification d'arbres de décision (voir section 2.2 page 43 et [26, 31]) impliquent un temps de calcul considérable et une utilisation de la mémoire qui peut avorter l'optimisation du MDP.

Dans le cas des ADD, les variables d'arité quelconques doivent d'abord être «transformées» en $\tilde{E}_{\mathbb{N}}(\log_2(n))$ variables binaires ($\tilde{E}_{\mathbb{N}}(p)$ est le plus petit entier supérieur ou égal à p). Or, la profondeur (resp. largeur) des ADD augmente linéairement (resp. exponentiellement) avec le nombre de variables, si bien que l'arité des variables a des effets similaires sur les arbres de décision et sur les ADD. Aussi, la résolution des MDP basés sur les ADD devient vite rédhibitoire lorsque le nombre de variables d'état est grand, comme le montre l'étude menée dans [31].

Ainsi, quelque soit le modèle utilisé, il est intéressant de réduire au maximum l'arité des variables d'état. Dans le cas des MDP énumérés, nous avons présenté dans la section 2.1 (page 29) une technique permettant de réduire la dimension de l'espace d'états en agrégeant les états au sein de partitions faiblement couplées. Nous proposons donc, dans cette partie, d'énumérer localement le MDP factorisé par rapport aux variables d'état dont nous souhaitons réduire l'arité. Nous présentons le mécanisme de remplacement de l'ancienne variable par la nouvelle variable abstraite, d'arité fortement réduite. Nous comparerons enfin le coût de réduction de l'arité des variables d'état au gain sur l'optimisation du MDP factorisé.

Plan de la partie II

Chapitre 3 : étude théorique de la réduction d'arité de certaines variables d'état, qui peut être considéré comme une généralisation des techniques de décomposition de MDP énumérés aux MDP factorisés

Chapitre 4 : proposition d'algorithme permettant de réduire automatiquement l'arité de certaines variables d'état, et de construire le MDP factorisé abstrait qui a les mêmes dynamique et utilités que le MDP factorisé initial

Chapitre 3

Hiérarchisation de variables d'état de MDP factorisés par décomposition de sous-MDP énumérés

3.1 Équivalence entre MDP énumérés et factorisés

Afin de réduire l'arité d'une variable d'état, nous souhaitons partitionner le sous-espace engendré par cette variable, de sorte à appliquer des techniques de décomposition de MDP énumérés sur ce sous-espace. Le MDP étant globalement factorisé, nous devons introduire la notion de sous-MDP énumérés, dont la dynamique et les récompenses sont celles du MDP factorisé global, restreintes au sous-espace engendré par la variable d'état réduite.

3.1.1 Modèle énuméré de MDP factorisé

Le théorème 7 permet de définir la représentation énumérée d'un MDP factorisé, de mêmes dynamique et utilités que le MDP factorisé initial. L'espace d'états énuméré est construit en instanciant toutes les variables d'états successivement, et les transitions sont obtenues en parcourant complètement les arbres de décision du MDP factorisé.

Démonstration. Le MDP énuméré ainsi défini est équivalent au MDP factorisé si ses états (resp. actions) sont en bijection avec ceux (resp. celles) du MDP factorisé,

Théorème 7 *Modèle énuméré de MDP factorisé*

Soit un MDP factorisé $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
La représentation énumérée de ce MDP est le MDP $\langle \mathcal{E}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$ *défini par :*

- $\mathcal{E} = \{\wedge_{i=1}^{n}(X_i = v_i) \ : \ v_i \in V_i \ , \ 1 \leqslant i \leqslant n\}$
- $\tilde{\mathcal{A}} = \mathcal{A}$
- $\tilde{\mathcal{T}} = \left(\tilde{T}^a\right)_{a \in \tilde{\mathcal{A}}}$ *tel que :*

$$\forall\, a \in \tilde{\mathcal{A}} \ , \ \tilde{T}^a \ : \ \begin{array}{lcl} \mathcal{E} \times \mathcal{E} & \longrightarrow & [0;1] \\ (\wedge_{i=1}^{n} v_i, \wedge_{i=1}^{n} v_i') & \longmapsto & \displaystyle\prod_{i=1}^{n} T_i^a \left(\wedge_{j=1}^{n} v_j, \wedge_{j \neq i} v_j'\right)(v_i') \end{array}$$

- $\tilde{\mathcal{R}} = \left(\tilde{R}^a\right)_{a \in \tilde{\mathcal{A}}}$ *tel que :*

$$\forall\, a \in \tilde{\mathcal{A}} \ , \ \tilde{R}^a \ : \ \begin{array}{lcl} \mathcal{E} \times \mathcal{E} & \longrightarrow & \mathbb{R} \\ (\wedge_{i=1}^{n} v_i, \wedge_{i=1}^{n} v_i') & \longmapsto & R^a \left(\wedge_{i=1}^{n} v_i, \wedge_{i=1}^{n} v_i'\right) \end{array}$$

et si ses transitions et ses récompenses sont égales à celles du MDP factorisé, à la bijection près entre les espaces d'état et d'action.

Les actions sont par définition identiques.
En ce qui concerne les états, $\mathcal{E}$ et $\mathcal{V}$ sont équipotents car ils ont le même nombre d'éléments. Ainsi, les récompenses des transitions sont équivalentes car les ensembles de définition des fonctions de récompense des deux modèles de MDP sont équipotents et que les valeurs de chacun de ces ensembles par les fonctions de récompense sont égales.
Il reste à démontrer que les dynamiques stochastiques des deux modèles de MDP sont équivalentes, c'est-à-dire que $\mathcal{T}$ et $\tilde{\mathcal{T}}$ définissent les mêmes transitions.
Pour une action $a \in \mathcal{A}$ fixée, la fonction de transition complète du MDP factorisé, qui représente la matrice des transitions entre les états, est (cf. section 2.2 page 43) :

$$f^a : \begin{array}{lcl} \mathcal{V} \times \mathcal{V} & \longrightarrow & \mathbb{R} \\ (\wedge_{i=1}^{n} v_i, \wedge_{i=1}^{n} v_i') & \longmapsto & \displaystyle\prod_{i=1}^{n} T_i^a(\wedge_{j=1}^{n} v_j, \wedge_{j \neq i} v_j')(v_i') \end{array}$$

Cela prouve que $\tilde{T}$ définit bien les transitions du MDP énuméré par définition de l'ensemble $\mathcal{E}$ des états énumérés.
De plus, l'ensemble de définition de f^a a un cardinal de $\prod_{i=1}^{n} |V_i|^2$, si bien que le nombre total (énuméré) de transitions du MDP factorisé est : $|\mathcal{A}| \cdot \prod_{i=1}^{n} |V_i|^2$, c'est-à-dire autant que $\tilde{\mathcal{T}}$. Ainsi, $\mathcal{T}$ et $\tilde{\mathcal{T}}$ représentent le même nombre de transitions (énumérées), par ailleurs égales, si bien qu'ils définissent les mêmes transitions. □

Illustrons le théorème précédent par un exemple simple. Soit un MDP factorisé à

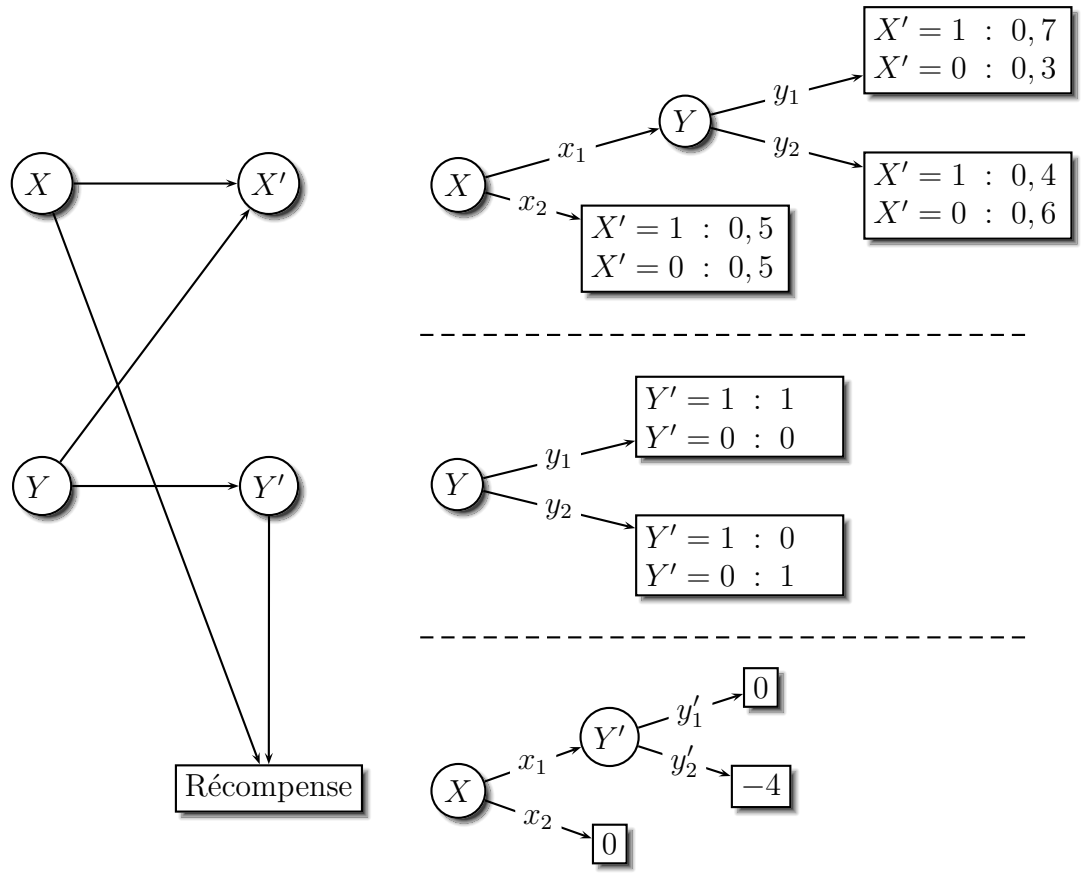


FIG. 3.1 – Exemple de modèle énuméré de MDP factorisé : DBN de l'action a_1 du modèle factorisé

deux variables X et Y et deux actions a_1 et a_2. Le DBN de l'action a_1 est représenté sur la figure 3.1 et celui de l'action a_2 sur la figure 3.2. Le modèle énuméré résultant est représenté sur la figure 3.3.

3.1.2 Modèle factorisé de MDP énuméré

La transformation inverse (cf. théorème 8) est plus simple, du moins si l'on n'analyse pas la structure de l'espace d'états énuméré afin d'en extraire des caractéristiques, qui définissent alors les variables du MDP factorisé [72]. Si ces caractéristiques ne sont pas connues, on ne peut définir qu'une seule variable d'état, qui caractérise simplement l'indice de l'état du MDP énuméré. La variable du MDP factorisé s'identifie donc à l'espace d'états du MDP énuméré. Cette transformation correspond à la transformation précédente lorsque le MDP factorisé n'a qu'une seule variable.

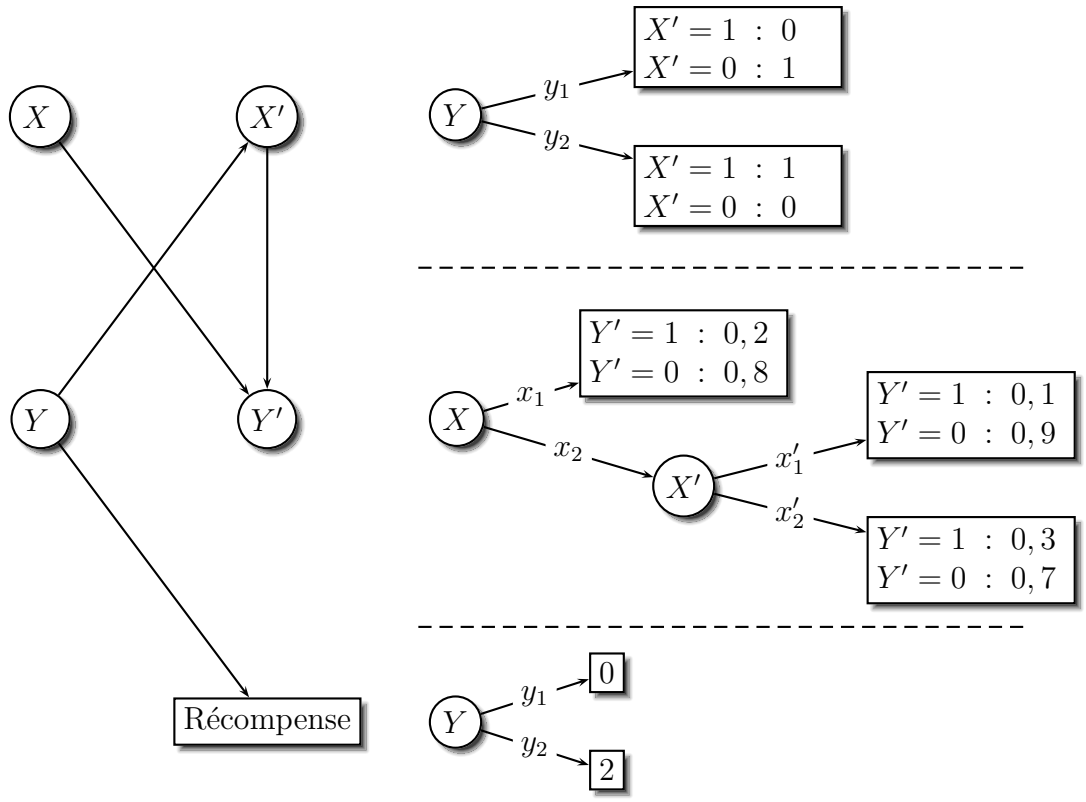


Fig. 3.2 – Exemple de modèle énuméré de MDP factorisé : DBN de l'action a_2 du modèle factorisé

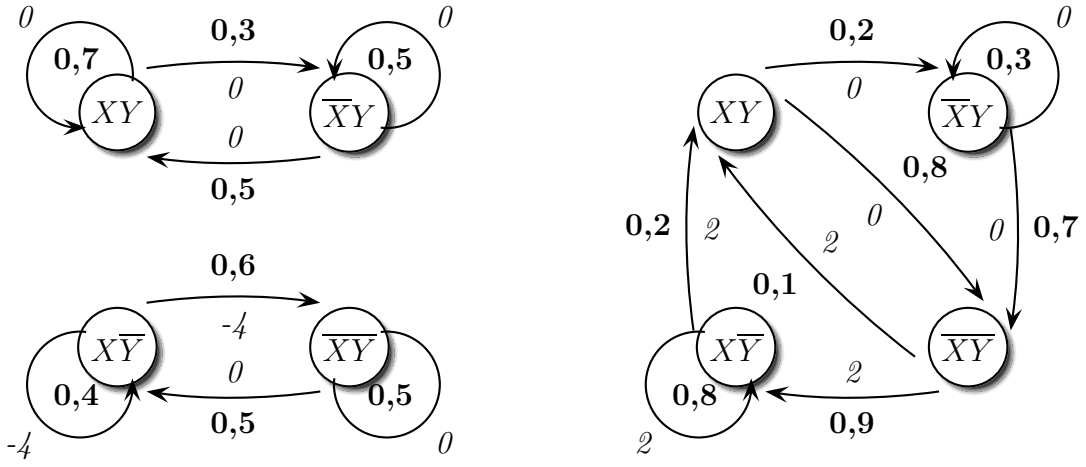
Théorème 8 *Modèle factorisé de MDP énuméré*

Soit un MDP énuméré $\langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
Une représentation factorisée de ce MDP est le MDP $\langle \mathcal{V}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$ *défini par :*

- $\mathcal{V} = \mathcal{E}$
- $\tilde{\mathcal{A}} = \mathcal{A}$
- $\tilde{\mathcal{T}} = \left(\tilde{T}^a\right)_{a \in \tilde{\mathcal{A}}}$ *tel que :*

$$\forall\, a \in \tilde{\mathcal{A}}\ ,\ \tilde{T}^a\ :\ \begin{array}{ccc} \mathcal{V} & \longrightarrow & \mathcal{F}(\mathcal{V}, [0;1]) \\ v & \longmapsto & \tilde{T}^a(v)\ :\ \begin{array}{ccc} \mathcal{V} & \to & [0;1] \\ v' & \mapsto & T^a(v, v') \end{array} \end{array}$$

- $\tilde{\mathcal{R}} = \mathcal{R}$



action a_1 action a_2

FIG. 3.3 – Exemple de modèle énuméré du MDP factorisé défini dans les figures 3.1 et 3.2 (les probabilités sont en gras et les récompenses sont en italiques)

Démonstration. Il est évident qu'il y a bijection entre $\mathcal{E}$ et $\mathcal{V}$, $\mathcal{A}$ et $\tilde{\mathcal{A}}$, $\mathcal{R}$ et $\tilde{\mathcal{R}}$. Il suffit donc de démontrer que les transitions des deux modèles de MDP sont égales.

Par définition des MDP factorisés (cf. définition 7 page 7), nous savons que $\tilde{T}^a(v)(v')$ représente la probabilité de l'événement "$X' = v'$" à un instant donné, sachant les événements a et "$X = v$" à l'instant précédent. Or, cette probabilité correspondant exactement à la définition de $T^a(v, v')$ pour les MDP énumérés (cf. définition 2 page 17).
De plus, la liste de fonctions $\tilde{\mathcal{T}}$ permet d'obtenir $(|\mathcal{A}|\cdot|\mathcal{V}|)\cdot|\mathcal{V}|$ transitions (énumérées puisqu'il n'y a qu'une seule variable d'état), c'est-à-dire autant que pour le MDP énuméré : $|\mathcal{A}| \cdot |\mathcal{E}|^2$.
Ainsi, les MDP énumérés et factorisés ont le même nombre de transitions, par ailleurs égales, si bien qu'ils définissent les mêmes transitions. □

Reprenons l'exemple du MDP énuméré obtenu dans la figure 3.3. Supposons que nous ne connaissons pas les caractéristiques X et Y du problème : nous ne pouvons donc décrire qu'une seule caractéristique V, dont les valeurs sont les indices des états du MDP énuméré. Pour cet exemple, V peut donc prendre comme valeurs XY, $\overline{X}Y$, $X\overline{Y}$ et $\overline{XY}$. Le modèle factorisé, décrit par V, du MDP énuméré défini dans la figure 3.3 est représenté dans les figures 3.4 (action a_1) et 3.5 (action a_2). Le MDP factorisé obtenu n'a pas du tout la même structure que le MDP factorisé initial (cf. figures 3.1 et 3.2), alors qu'ils sont pourtant équivalents d'après les théorèmes 8 et 7. Les DBN du MDP factorisé initial sont même plus compacts grâce à la factorisation en deux variables d'état au lieu d'une seule pour l'autre MDP factorisé. Ceci reflète

l'importance du choix et de la mise en évidence des caractéristiques du problème dans la modélisation du MDP factorisé.

3.1.3 Application à la réduction d'arité des variables d'état d'un MDP factorisé

Afin de réduire l'arité d'une variable d'état X donnée, nous souhaitons énumérer localement le MDP factorisé, ce qui peut être réalisé au moins par les deux méthodes extrêmes suivantes :

- soit en énumérant le sous-MDP factorisé défini par la variable X et les variables reliées à X dans les DBN du MDP factorisé (M1) ;
- soit en fixant toutes les variables dont dépend X' dans les DBN du MDP factorisé, et en énumérant les sous-MDP définis pour chaque instanciation de ces variables (M2).

Le premier cas nécessite l'utilisation du théorème 7. La décomposition du sous-MDP énuméré donne une macro-variable qui combine X et les variables reliées à X dans les DBN du MDP factorisé. Le deuxième cas requière l'application du théorème 8, puisque les sous-MDP locaux ne sont définis que par une seule même variable. La décomposition des sous-MDP énumérés mène toujours à la même macro-variable $\tilde{X}$. Ce dernier théorème est certes plus simple, mais il doit être utilisé autant de fois qu'il y a de façons possibles d'instancier les variables d'état en relation avec X dans les DBN du MDP factorisé. Toutefois, la macro-variable obtenue dans le deuxième cas (partitions des valeurs de X) est plus simple à interpréter que dans le premier cas (partitions des valeurs de plusieurs variables combinées).

Une méthode intermédiaire pour réduire l'arité d'une variable consiste à utiliser (M1), puis à re-factoriser localement le MDP énuméré obtenu par rapport à des caractéristiques qui augmentent localement la compacité du MDP factorisé (M3). En effet, nous avons vu plus haut dans ce même chapitre que la compacité des DBN dépend beaucoup des caractéristiques qui décrivent l'espace d'états du MDP.

Le tableau 3.1 indique les avantages et inconvénients de ces trois méthodes. La mise en œuvre de la méthode (M3) reposerait sur l'extraction automatique de caractéristiques *efficaces* en terme de compacité des DBN du MDP factorisé, ce qui nous semble ardu et qui, à notre connaissance, n'a pas été proposé dans la littérature. Il existe bien des méthodes permettant d'extraire automatiquement des caractéristiques de l'espace d'états [72], mais nous n'avons aucune garantie que ces nouvelles caractéristiques sont plus compactes que celles du MDP factorisé initial.

Au contraire, les méthodes (M1) et (M2) se basent sur la *décomposition* du MDP résultant énuméré, qui est une technique réputée pour son efficacité quant à la réduction du nombre d'états énumérés. Cependant, la complexité de la décomposition de MDP énuméré [23, 24] dépend fortement de la taille du MDP énuméré et de la

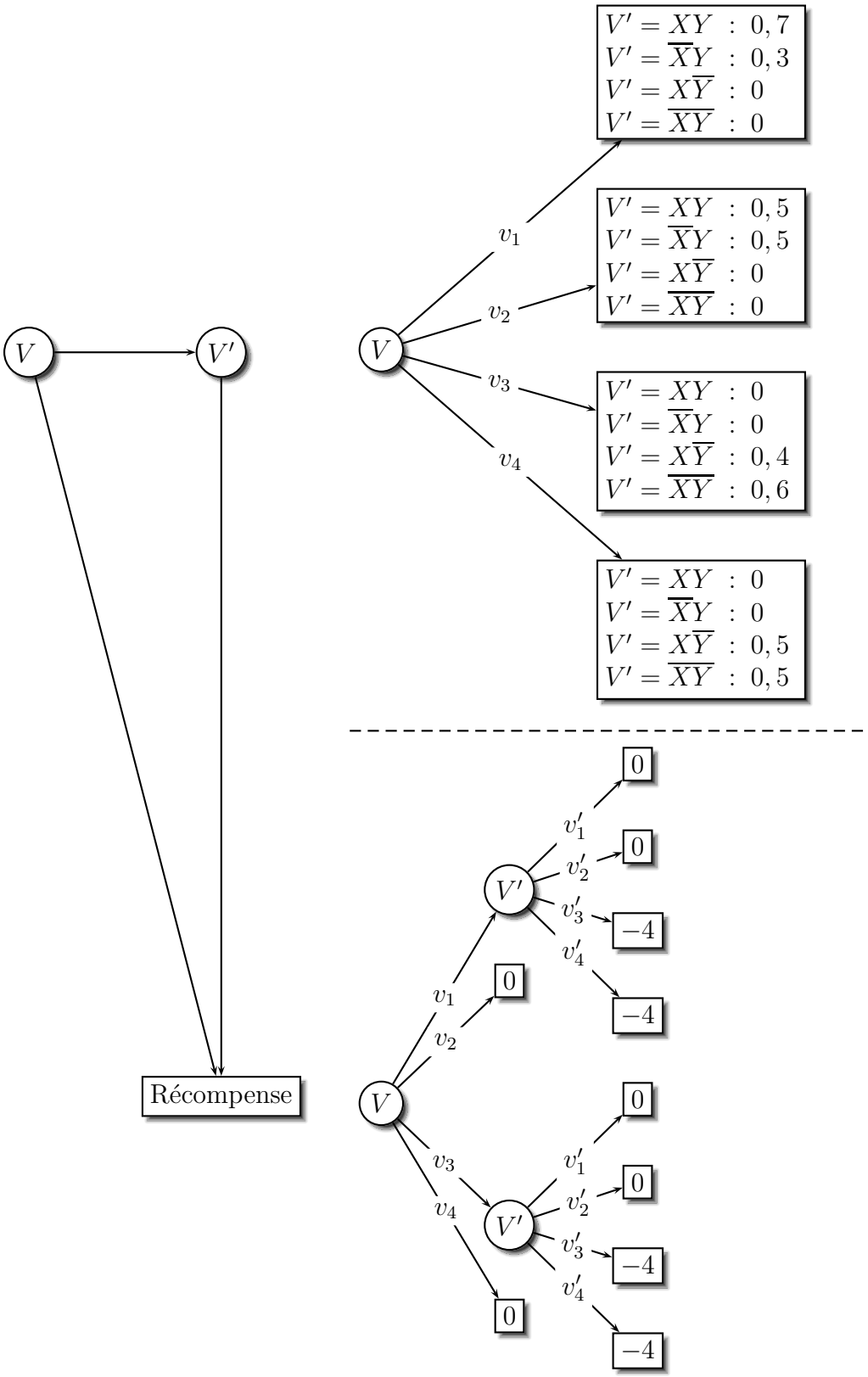


FIG. 3.4 – Exemple de modèle factorisé de MDP énuméré (cf. figure 3.3) : DBN de l'action a_1 du modèle factorisé

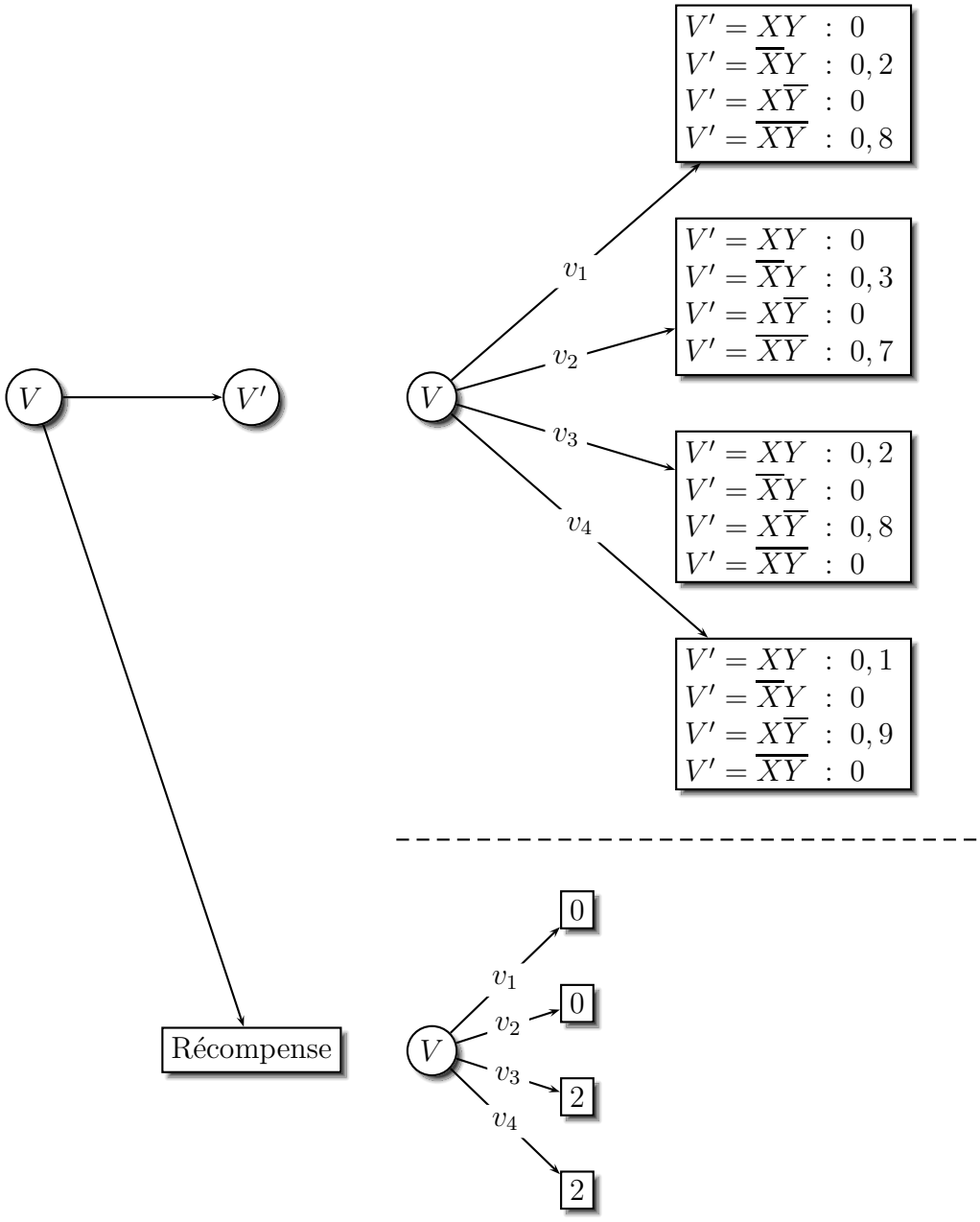


Fig. 3.5 – Exemple de modèle factorisé de MDP énuméré (cf. figure 3.3) : DBN de l'action a_2 du modèle factorisé

Méthode	Avantages	Inconvénients
M1	décomposition du MDP énuméré obtenu (efficacité)	MDP énuméré potentiellement grand et non nécessairement structuré en régions faiblement couplées
M2	décomposition des MDP énumérés obtenus (efficacité), faible taille des MDP énumérés	plusieurs décompositions nécessaires (une par MDP énuméré)
M3	pas de décomposition du MDP énuméré obtenu (simplicité)	aucune méthode simple connue pour extraire automatiquement des caractéristiques qui augmentent la compacité des DBN

TAB. 3.1 – Comparaison de méthodes pour réduire l'arité de variables d'état

structure du graphe d'états (mise en évidence de régions faiblement couplées).

Or, contrairement à la méthode (M2) pour laquelle il est relativement aisé de choisir des variables dont les valeurs énumérées sont structurées en régions faiblement couplées (variable «position géographique» par exemple), la méthode (M1) a l'inconvénient de générer un MDP énuméré dont la structure est difficilement prévisible, car l'espace d'états énuméré est le produit de différentes variables d'états. En outre, la méthode (M2) impose certes de décomposer possiblement un grand nombre de MDP énumérés, mais chacun de ces MDP énumérés est de petite taille. Pour ces raisons, nous avons choisi d'implémenter la méthode (M2), que nous développons dans la suite.

3.2 Sous-MDP énumérés par rapport à une variable d'état

3.2.1 X-dépendance

Considérons un MDP factorisé dont nous souhaitons réduire l'arité d'une variable d'état X. Le sous-MDP énuméré par rapport à la variable X *uniquement*, c'est-à-dire tel que l'espace d'états correspond aux différentes valeurs de X, ne doit contenir que des transitions du type $P(X' \mid a, X)$ (méthode M2). Ceci est possible à condition de fixer les valeurs de toutes les variables dont dépend X' dans les DBN du MDP factorisé. Nous appelons *X-dépendance* l'ensemble des variables d'état dont dépend X' (cf. définition 9). Cet ensemble correspond à toutes les variables qui sont reliées

à X' dans la représentation graphique des DBN du MDP factorisé.

Définition 9 X-dépendance

Soit un MDP factorisé $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
La X-dépendance est l'ensemble $\mathcal{D}(X)$ défini par :

$$Y \in \mathcal{D}(X) \iff \begin{cases} Y \in \mathcal{V} \cup (\mathcal{V}' \setminus X') \\ \exists\, a \in \mathcal{A}\ ,\ \exists\, (\alpha, \beta) \in V_Y^2\ ,\ \alpha \neq \beta\ : \\ \qquad T_X^a(\ldots, Y = \alpha, \ldots) \neq T_X^a(\ldots, Y = \beta, \ldots) \end{cases}$$

Illustrons la définition précédente par un exemple simple de MDP factorisé ayant une seule action et trois variables X, Y et Z, défini dans la figure 3.6. Cet exemple servira dans la suite à décrire l'intégration de la macro-variable issue de la décomposition des valeurs de X dans le DBN du MDP factorisé. Dans cet exemple, la X-dépendance est l'ensemble $\{X, Z, Y'\}$.

Revenons au cadre général. Dans le pire des cas, il existe une action pour laquelle X' dépend de toutes les variables pré-action $Y \in \mathcal{V}$, et des variables post-actions $Y' \neq X'$. Le nombre de sous-MDP énumérés à construire en fixant successivement toutes ces variables est donc, dans le pire des cas, de $|V_X| \cdot \prod_{Y \neq X} |V_Y|^2$. Néanmoins, X a certes un grand nombre de valeurs, mais ce nombre est en général faible comparé à la taille critique des espaces d'états pour la décomposition de MDP énumérés (plusieurs centaines d'états). Ainsi, dans la majorité des problèmes, chaque sous-MDP énuméré est facile à décomposer.

Puisque la X-dépendance correspond aux variables d'état dont dépend l'arbre de transitions de la variable post-action X', cette notion peut être naturellement étendue à l'arbre de récompense de chaque action. Ainsi, la $\mathcal{R}$-dépendance est l'ensemble des variables d'état dont dépendent les arbres de récompense, toutes actions confondues.

3.2.2 Énumération des sous-MDP par rapport à X en parcourant la X-dépendance et la $\mathcal{R}$-dépendance

En fixant les valeurs des variables de la X-dépendance et de la $\mathcal{R}$-dépendance, les transitions du MDP factorisé peuvent être restreintes au sous-espace V_X engendré par la variable X. Ainsi, comme l'indique la définition 10, nous pouvons construire autant de sous-MDP énumérés qu'il est possible d'évaluer simultanément les variables de la X-dépendance et de la $\mathcal{R}$-dépendance conjuguées. Comme nous le préciserons plus loin, les dynamiques de ces sous-MDP supposent que les variables d'états différentes de X ne varient quasiment pas comparées à X.

Considérons par exemple le MDP factorisé à 1 action et 3 variables d'état, défini dans la figure 3.6. Supposons que nous souhaitions réduire l'arité de la variable X en

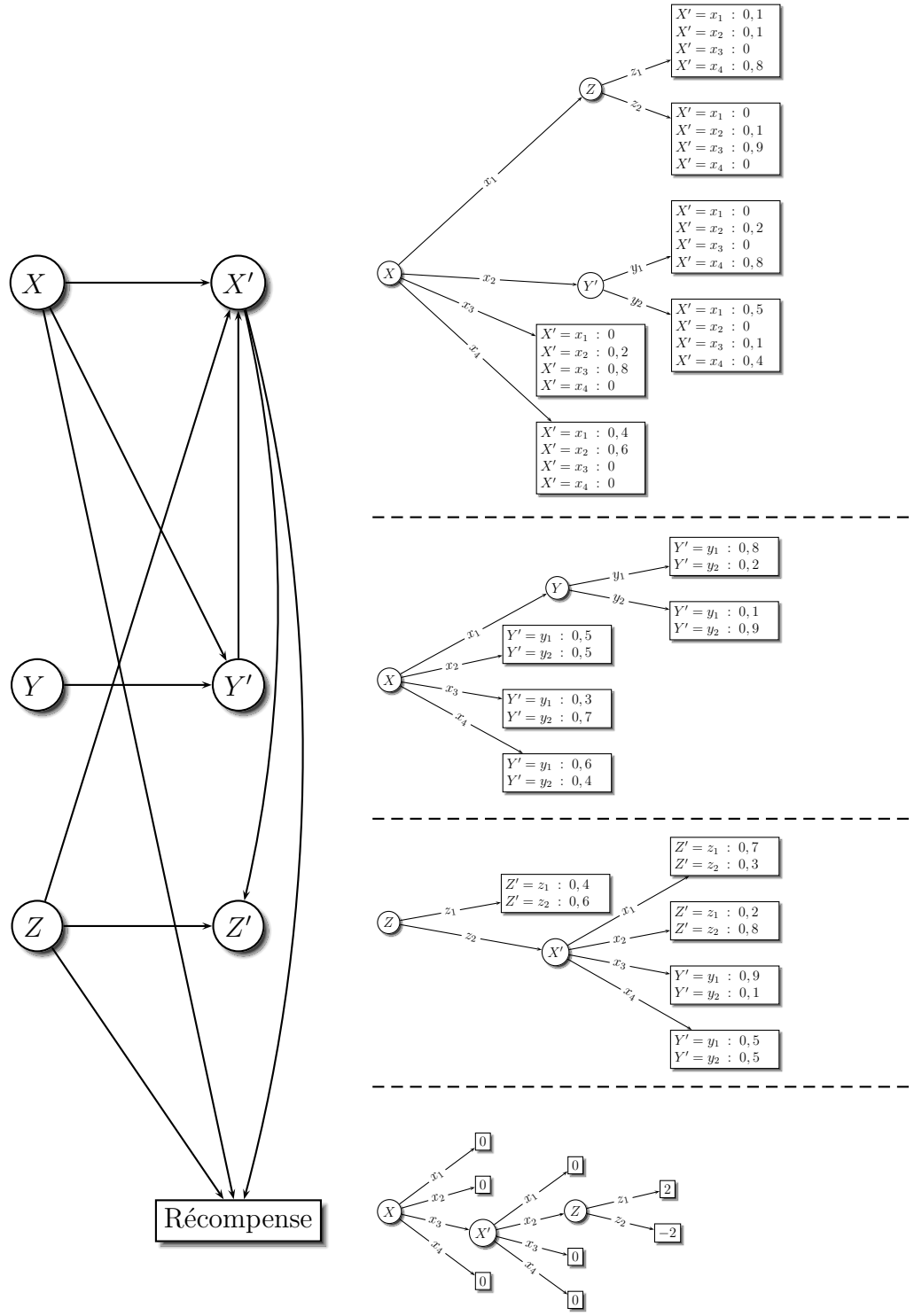


FIG. 3.6 – DBN d'un MDP factorisé à 1 action et 3 variables d'état

Définition 10 Sous-MDP énumérés par rapport à une variable d'état

Soit un MDP factorisé $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
Soit $1 \leqslant p \leqslant n$ l'indice d'une variable de grande arité.
Soient $\mathcal{D}(X_p) \setminus X_p$ la X_p-dépendance privée de X_p, et $\mathcal{D}(\mathcal{R}) \setminus \{X_p, X'_p\}$ la $\mathcal{R}$-dépendance privée de X_p et X'_p . Le produit cartésien de ces deux dépendances peut se noter $V_{i_1} \times \cdots \times V_{i_q} \times V_{j_1} \times \cdots \times V_{j_r}$ en séparant les variables pré-action et post-action.
Pour chaque état $\wedge_{k=1}^{q} v_{i_k} \wedge_{k=1}^{r} v'_{j_k}$ de $(\mathcal{D}(X_p) \setminus X_p) \times \big(\mathcal{D}(\mathcal{R}) \setminus \{X_p, X'_p\}\big)$, les sous-MDP énumérés par rapport à la variable d'état X_p sont les MDP $\langle V_p, \mathcal{A}, \tilde{\mathcal{T}}_{\wedge_{k=1}^{q} v_{i_k} \wedge_{k=1}^{r} v'_{j_k}}, \tilde{\mathcal{R}}_{\wedge_{k=1}^{q} v_{i_k} \wedge_{k=1}^{r} v'_{j_k}} \rangle$ dont les transitions sont définies par :

– $\forall\, a \in \mathcal{A}$, $\tilde{T}^a_{\wedge_{k=1}^{q} v_{i_k} \wedge_{k=1}^{r} v'_{j_k}} : \begin{array}{ccc} V_p \times V_p & \longrightarrow & [0;1] \\ (e, e') & \longmapsto & T^a_p(v_{i_1}, \ldots, v_{i_q}, e, v'_{j_1}, \ldots, v'_{j_r})(e') \end{array}$

– $\forall\, a \in \mathcal{A}$, $\tilde{R}^a_{\wedge_{k=1}^{q} v_{i_k} \wedge_{k=1}^{r} v'_{j_k}} : \begin{array}{ccc} V_p \times V_p & \longrightarrow & \mathbb{R} \\ (e, e') & \longmapsto & R^a(v_{i_1}, \ldots, v_{i_q}, e, v'_{j_1}, \ldots, v'_{j_r}, e') \end{array}$

décomposant les sous-MDP énumérés par rapport à X. Dans cet exemple, l'union de la X-dépendance privée de X et de la $\mathcal{R}$-dépendance privée de X et X' est l'ensemble $V_Z \times V_{Y'}$. Le nombre de sous-MDP à énumérer par rapport à X est donc de 4 puisque Z et Y' ont 2 valeurs chacune. Dans le pire des cas, il faudrait énumérer $2^4 = 16$ sous-MDP, en considérant les variables Y, Y', Z et Z'. Cependant, ce cas est très rare dans la plupart des problèmes. Les sous-MDP énumérés sont représentés dans la figure 3.7. Leurs espaces d'états sont identiques, mais leurs structures (transitions) sont différentes. Ainsi, le partitionnement automatique de leurs espaces d'états énumérés risque de mener à des macro-états différents pour chaque sous-MDP. Afin de traiter simultanément les structures de chaque sous-MDP, nous proposons de construire un seul sous-MDP énuméré, appelé «MDP sous-jacent de la variable d'état X».

3.3 MDP sous-jacent d'une variable d'état

Considérons les sous-MDP énumérés par rapport à la variable d'état X, précédemment définis. La définition 11 spécifie un seul sous-MDP énuméré dont l'ensemble des transitions est l'union des transitions de chaque sous-MDP énuméré par rapport aux variables de la X-dépendance et de la $\mathcal{R}$-dépendance. Ceci nécessite donc de dupliquer les actions autant de fois qu'il est nécessaire d'énumérer de sous-MDP, pour chaque valeur des variables précédentes. Ainsi, pour le MDP factorisé représenté dans la figure 3.6, le MDP sous-jacent de la variable X est un MDP énuméré à 4 états et 4 actions, dont les transitions pour chaque action sont définies dans la figure 3.7 (chaque action correspond à 1 des 4 sous-MDP énumérés).

Le MDP énuméré ainsi défini tient compte simultanément de toutes les transi-

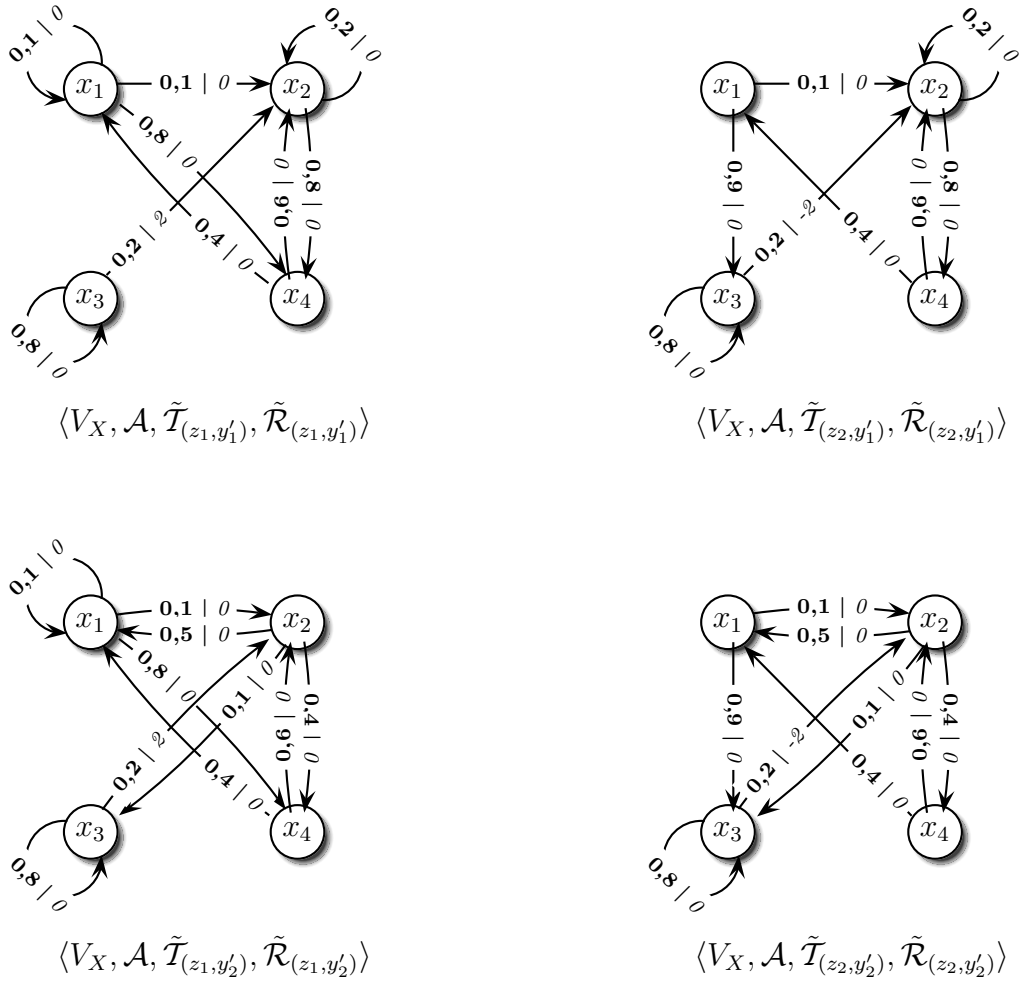


FIG. 3.7 – Sous-MDP énumérés par rapport à la variable X pour le MDP factorisé défini dans la figure 3.6 (probabilités en gras et récompenses en italique)

tions des sous-MDP énumérés par rapport à la variable d'état X. Il est alors possible de partitionner son ensemble d'états à l'aide d'un algorithme de partitionnement automatique tel celui de [25]. Un tel algorithme prendra donc en compte toutes les structures différentes de chaque sous-MDP énuméré, afin d'optimiser le partitionnement de l'espace d'états sur toutes ces structures.

Définition 11 MDP sous-jacent d'une variable d'état

Soit un MDP factorisé $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
Soit $1 \leqslant p \leqslant n$ l'indice d'une variable d'état de grande arité.
Soit $N = |\left(\mathcal{D}(X_p) \setminus X_p\right) \times \left(\mathcal{D}(\mathcal{R}) \setminus \{X_p, X'_p\}\right)|$ le nombre de sous-MDP énumérés par rapport à X_p.
Soit $\langle V_p, \mathcal{A}, \tilde{\mathcal{T}}_i, \tilde{\mathcal{R}}_i \rangle_{1 \leqslant i \leqslant N}$ l'ensemble des sous-MDP énumérés par rapport à X_p .
Le MDP sous-jacent de la variable X_p est le MDP énuméré $\langle V_p, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$, défini par :

- $\tilde{\mathcal{A}} = \mathcal{A} \times [1; N]$
- $\tilde{\mathcal{T}} = \left(\tilde{T}^{(a,i)}\right)_{(a,i) \in \tilde{\mathcal{A}}}$ tel que :

$$\forall\ (a,i) \in \tilde{\mathcal{A}}\ ,\ \tilde{T}^{(a,i)}\ :\ \begin{array}{ccc} V_p \times V_p & \longrightarrow & [0;1] \\ (e,e') & \longmapsto & \tilde{T}_i^a(e,e') \end{array}$$

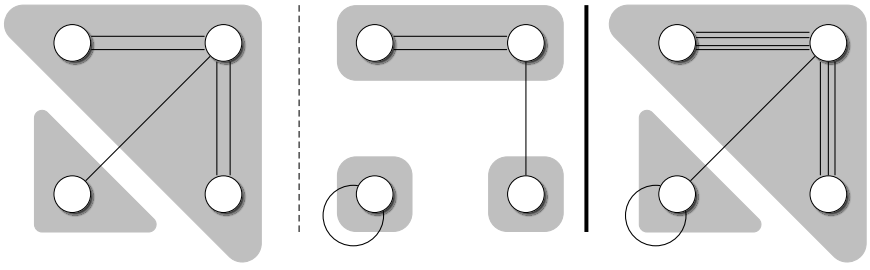
- $\tilde{\mathcal{R}} = \left(\tilde{R}^{(a,i)}\right)_{(a,i) \in \tilde{\mathcal{A}}}$ tel que :

$$\forall\ (a,i) \in \tilde{\mathcal{A}}\ ,\ \tilde{R}^{(a,i)}\ :\ \begin{array}{ccc} V_p \times V_p & \longrightarrow & \mathbb{R} \\ (e,e') & \longmapsto & \tilde{R}_i^a(e,e') \end{array}$$

3.4 Partitionnement du MDP sous-jacent et décomposition des sous-MDP énumérés

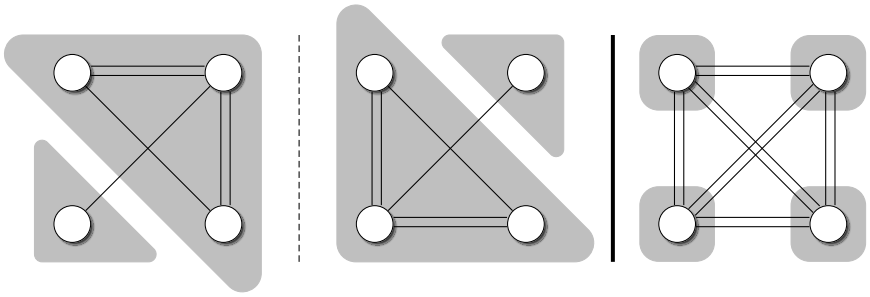
Le partitionnement du MDP sous-jacent de X est d'autant plus efficace que les structures des différents sous-MDP énumérés sont proches. Dans le cas contraire, les différents partitionnements obtenus pour chaque sous-MDP énuméré — s'ils devaient être traités séparément — risquent de se recouvrir, au point que le partitionnement final peut être très fragmenté. La figure 3.8 illustre un cas favorable et un cas défavorable de décomposition du MDP sous-jacent de X. Dans le premier cas, l'espace d'états est réduit de moitié alors que dans le second cas, l'espace d'états reste inchangé. En effet, pour ce dernier cas, il n'existe pas de connection entre deux états qui ait moins de transitions que les connections entre les autres états. Ainsi, il est impossible de différencier un groupe d'états d'autres groupes d'états. Le partitionnement du MDP sous-jacent ne présente dans ce cas aucun intérêt pour la réduction de l'arité de la variable d'état X.

Alors que le partitionnement du MDP sous-jacent a une signification topologique, celle de trouver un partitionnement commun et optimal *en moyenne* pour tous les sous-MDP énumérés, la décomposition du MDP sous-jacent n'a en revanche aucun sens physique. En effet, les transitions du MDP sous-jacent proviennent de sous-



Sous-MDP énuméré $\mathcal{P}_1$ Sous-MDP énuméré $\mathcal{P}_2$ MDP sous-jacent

Cas favorable



Sous-MDP énuméré $\mathcal{P}_1$ Sous-MDP énuméré $\mathcal{P}_2$ MDP sous-jacent

Cas défavorable

FIG. 3.8 – Cas favorable (2 macro-états) et cas défavorable (4 macro-états) de partitionnement du MDP sous-jacent à X dont les transitions sont représentées dans la figure 3.7

MDP énumérés différents qui correspondent chacun à des instanciations différentes de certaines variables d'état. Considérons par exemple le MDP factorisé défini dans la figure 3.6 et dont les sous-MDP énumérés par rapport à X sont représentés dans la figure 3.7. Une trajectoire possible dans le MDP sous-jacent à X, en partant de $X = x_1$ est :

$$x_1 \xrightarrow{(a,3)\triangleq(z_1,y_2')} x_4 \xrightarrow{(a,3)\triangleq(z_1,y_2')} x_2 \xrightarrow{(a,1)\triangleq(z_1,y_1')} x_4$$

Cette trajectoire n'a a priori aucun sens physique puisqu'elle suppose un changement de la variable Y' dont le modèle de transition stochastique n'est pas intégré dans le MDP sous-jacent de X. Intégrer le modèle de transitions stochastiques de Y' dans le MDP sous-jacent de X reviendrait à rajouter Y' dans l'espace d'états du MDP sous-jacent.

La décomposition du MDP sous-jacent de X générerait des politiques locales dont les trajectoires n'auraient pas de sens physique. En revanche, la décomposition de chaque sous-MDP énuméré, **pour un partitionnement identique à tous les sous-MDP énumérés**, produit des politiques locales qui correspondent à des valeurs constantes des variables de la X-dépendance et de la $\mathcal{R}$-dépendance (privées de X et X'). La décomposition des sous-MDP énumérés est donc physiquement possible à condition que ces variables varient peu entre deux instants du processus de décision, comparées à X (définition 12). Cette hypothèse consiste à relaxer les dépendances stochastiques entre X et les autres variables d'état, ce que nous qualifierons de *découplage* entre la variable X et les variables dont elle dépend.

Définition 12 Hypothèse de découplage

L'hypothèse de découplage signifie que les variables d'état de la X-dépendance et de la $\mathcal{R}$-dépendance privées de X et X' varient peu par rapport à X entre deux instants du processus de décision :

$$\exists\, \epsilon > 0 \ : \ \forall\, Y \in (\mathcal{D}(X) \cup \mathcal{D}(\mathcal{R})) \setminus \{X, X'\}\ ,\ P(Y' \neq Y) < \epsilon P(X' \neq X)$$

En pratique, cette hypothèse n'entraîne pas de perte d'optimalité pour la stratégie optimisée, car une politique locale n'est plus suivie dès qu'une variable de la X-dépendance et de la $\mathcal{R}$-dépendance change de valeur. Autrement dit, une politique locale est toujours appliquée dans la configuration pour laquelle elle a été optimisée (valeurs constantes données des variables de la X-dépendance et de la $\mathcal{R}$-dépendance). Les stratégies locales qui intègrent des changements de valeurs des variables autres que X sont implicitement prises en compte, en considérant qu'elles sont chacune une union de politiques locales optimisées pour des valeurs fixées des variables autres que X. Dans ce contexte, l'hypothèse de découplage garantie d'empêcher des changements intempestifs de politiques locales à l'intérieur de chaque région, à chaque fois qu'une variable différente de X change de valeur.

3.5 MDP abstrait factorisé

Le partitionnement du MDP sous-jacent de X permet d'obtenir une variable d'état abstraite $\tilde{X}$ dont les valeurs correspondent à chaque partition du MDP sous-jacent de X. Les politiques locales générées pour chaque sous-MDP énuméré fournissent les macro-transitions entre les différentes valeurs de $\tilde{X}$ en supposant que les variables dont dépend X et la fonction de récompense sont constantes durant l'exécution de chaque politique locale. Sous cette hypothèse, il est possible de remplacer la variable d'état X par la variable abstraite $\tilde{X}$ dans le MDP factorisé initial,

sans modifier la dynamique markovienne et le modèle de récompense du processus décisionnel.

Le théorème 9 donne la définition du MDP abstrait factorisé obtenu en remplaçant la variable X par son abstraction $\tilde{X}$, et les actions par les macro-actions générées dans chaque sous-MDP énuméré par rapport à X. Ces macro-actions n'étant définies chacune que pour une seule instanciation des variables de la X_p-dépendance et de la $\mathcal{R}$-dépendance privées de X_p et X'_p, leurs transitions ne sont définies que sur le sous-espace d'états correspondant à l'instanciation de ces variables. Ceci explique la nullité des transitions et des récompenses sur le sous-espace d'états complémentaires. En particulier, les fonctions de transition ne représentent pas des probabilités sur le sous-espace complémentaire, puisque la somme des probabilités partant d'un état donné pour une macro-action donnée est nulle (cf. théorème 9).

Aussi, un « arbre de masque » doit être utilisé afin de restreindre l'opérateur de Bellman au sous-espace où chaque action est définie (cf. section 2.2 page 43). Cette méthode est plus efficace que de définir des probabilités NO-OP (les variables ne changent pas de valeur) sur les états où les actions ne sont pas définies. En effet, le filtrage des matrices de transition, représentées sous forme d'ADD, par les masques d'application des différentes actions, représentés sous forme de BDD, est très efficace car l'opérateur de Bellman agit alors sur des ADD de très petites tailles [69].

Il en résulte que **les matrices de transition du MDP factorisé abstrait sont très creuses**. Aussi, à la vue du théorème 9, la construction et l'optimisation du MDP abstrait factorisé peuvent paraître fastidieuses et complexes, mais elles sont en fait très efficaces en pratique. Les macro-actions sont nombreuses, mais **elles opèrent chacune sur un sous-espace d'états très petit**.

Démonstration. Les actions du MDP factorisé initial ne sont pas définies sur $\tilde{\mathcal{V}}$ car elles ne s'appliquent pas sur la macro-variable $\tilde{X}_p$. En effet, $\tilde{X}_p$ représente les macro-états du MDP sous-jacent de X_p et les transitions entre ces macro-états sont définies pour les politiques locales des sous-MDP énumérés décomposés. Ainsi, l'espace d'actions du MDP abstrait factorisé est l'union des politiques locales générées pour toutes les régions de chaque sous-MDP énuméré. Or il existe autant de sous-MDP énumérés que d'états de la X_p-dépendance et de la $\mathcal{R}$-dépendance privées de X_p et X'_p, soit $|\mathcal{K}|$ sous-MDP énumérés. L'espace d'actions abstrait est donc l'union des espaces d'actions définis pour chaque état de l'espace précédent.

Calculons les macro-probabilités conditionnelles de transition entre les variables du MDP abstrait factorisé. Il est raisonnable de traiter la variable abstraite $\tilde{X}_p$ à part, puisque les macro-actions ont été générées à partir des valeurs de cette variable.

Soient une macro-action π (politique locale d'un sous-MDP énuméré) et des valeurs fixées des variables d'état :

$$X_1 = v_1, \ldots, X_{p-1} = v_{p-1}, \tilde{X}_p = \tilde{v}_p, X_{p+1} = v_{p+1}, \ldots, X_n = v_n$$

Théorème 9 *MDP abstrait factorisé*

Soit un MDP factorisé $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.
Soit $1 \leqslant p \leqslant n$ *l'indice d'une variable d'état de grande arité.*
Soient $\tilde{X}_p$ *la variable d'état abstraite issue du partitionnement du MDP sous-jacent de* X_p *et* $\mathcal{K} = (\mathcal{D}(X_p) \setminus X_p) \times \left(\mathcal{D}(\mathcal{R}) \setminus \{X_p, X'_p\}\right)$ *la* X_p*-dépendance et la* $\mathcal{R}$*-dépendance privées de* X_p *et* X'_p.
Soit $\langle \tilde{V}_p, \tilde{\mathcal{A}}_e, \tilde{\mathcal{T}}_e, \tilde{\mathcal{R}}_e \rangle_{e \in \mathcal{K}}$ *l'ensemble des sous-MDP abstraits énumérés par rapport à* X_p *, issus de la décomposition des sous-MDP énumérés par rapport à* X_p *pour le partitionnement* $\tilde{X}_p$.
Le MDP abstrait factorisé $\langle \tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$ *, correspondant au MDP factorisé* $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *où la variable* X_p *est remplacée par son abstraction* $\tilde{X}_p$ *, est défini par :*

- $\tilde{\mathcal{V}} = \left(\bigotimes_{i \neq p} V_i \right) \times \tilde{V}_p$
- $\tilde{\mathcal{A}} = \bigcup_{e \in \mathcal{K}} \tilde{\mathcal{A}}_e$

  Soit $\Pi : \tilde{\mathcal{A}} \to \mathcal{K}$ *la fonction qui à une macro-action fait correspondre l'instanciation des variables de* $\mathcal{K}$ *du sous-MDP énuméré pour laquelle cette macro-action est définie.*
- $\tilde{\mathcal{T}} = \left(\tilde{T}_i^{\pi} \right)_{\substack{\pi \in \tilde{\mathcal{A}} \\ 1 \leqslant i \leqslant n}}$ *avec :*
  - $\forall\, \pi \in \tilde{\mathcal{A}}$, $\tilde{T}_p^{\pi} : \left(\bigotimes_{j \neq p} V_j \right)^2 \times \tilde{V}_p \longrightarrow \mathcal{F}(\tilde{V}_p, [0,1])$ *définie par :*

$$\tilde{T}_p^{\pi} \left(\bigwedge_{j \neq p} v_j, \bigwedge_{j \neq p} v'_j, \tilde{v}_p \right) (\tilde{v}'_p) = \begin{cases} 0 \text{ si } \bigwedge\limits_{j \neq p} v_j \bigwedge\limits_{j \neq p} v'_j \not\subset \Pi(\pi) \\ \tilde{\mathcal{T}}^{\pi}_{\bigwedge\limits_{j \neq p} v_j \bigwedge\limits_{j \neq p} v'_j} (\tilde{v}_p, \tilde{v}'_p) \text{ sinon} \end{cases}$$

  - $\forall\, \pi \in \tilde{\mathcal{A}}$, $\forall\, i \neq p$, $\tilde{T}_i^{\pi} : \left(\bigotimes_{j \neq p} V_j \right) \times \left(\bigotimes_{j \neq \{i,p\}} V_j \right) \times \tilde{V}_p^2 \longrightarrow \mathcal{F}(V_i, [0,1])$ *définie par :*

$$\tilde{T}_i^{\pi} \left(\bigwedge_{j \neq p} v_j, \bigwedge_{j \neq \{i,p\}} v'_j, \tilde{v}_p, \tilde{v}'_p \right) (v'_i) =$$
$$\begin{cases} 0 \text{ si } \bigwedge\limits_{j \neq p} v_j \bigwedge\limits_{j \neq p} v'_j \not\subset \Pi(\pi) \\ \dfrac{1}{|\tilde{v}_p|} \sum\limits_{v_p \in \tilde{v}_p} \dfrac{\sum\limits_{v'_p \in \tilde{v}'_p} \mathcal{T}_i^{\pi} \left(\bigwedge\limits_{1 \leqslant j \leqslant n} v_j, \bigwedge\limits_{j \neq i} v'_j \right) (v'_i) \cdot \mathcal{T}^{\pi}_{\bigwedge\limits_{j \neq p} v_j \bigwedge\limits_{j \neq p} v'_j} (v_p, v'_p)}{\sum\limits_{v'_p \in \tilde{v}'_p} \sum\limits_{v'_{i,k} \in V_i} T_i^{\pi} \left(\bigwedge\limits_{1 \leqslant j \leqslant n} v_j, \bigwedge\limits_{j \neq i} v'_j \right) \left(v'_{i,k} \right) \cdot \mathcal{T}^{\pi}_{\bigwedge\limits_{j \neq p} v_j \bigwedge\limits_{j \neq p} v'_j} (v_p, v'_p)} \text{ sinon} \end{cases}$$

- $\tilde{\mathcal{R}} = \left(\tilde{R}^{\pi} \right)_{\pi \in \tilde{\mathcal{A}}}$ *avec :*

$$\forall\, \pi \in \tilde{\mathcal{A}}\,,\ \tilde{R}^{\pi} : \begin{array}{ccc} \left(\bigotimes\limits_{j \neq p} V_j \right)^2 \times \tilde{V}_p^2 & \longrightarrow & \mathbb{R} \\ \left(\bigwedge\limits_{j \neq p} v_p, \bigwedge\limits_{j \neq p} v'_p, \tilde{v}_p, \tilde{v}'_p \right) & \longmapsto & \begin{cases} 0 \text{ si } \bigwedge\limits_{j \neq p} v_j \bigwedge\limits_{j \neq p} v'_j \not\subset \Pi(\pi) \\ \tilde{R}^{\pi}_{\bigwedge\limits_{j \neq p} v_j \bigwedge\limits_{j \neq p} v'_j} (\tilde{v}_p, \tilde{v}'_p) \text{ sinon} \end{cases} \end{array}$$

$$X'_1 = v'_1, \ldots, X'_{p-1} = v'_{p-1}, \tilde{X}'_p = \tilde{v}'_p, X'_{p+1} = v'_{p+1}, \ldots, X'_n = v'_n$$

L'arbre de transition de la macro-variable $\tilde{X}_p$ représente la fonction suivante :

$$\tilde{T}_p^{\pi}\left(\bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p\right)(\tilde{v}'_p) = P\left(\tilde{X}'_p = \tilde{v}'_p \mid \pi, \tilde{X}_p = \tilde{v}_p, \bigwedge_{j\neq p} X_j = v_j, \bigwedge_{j\neq p} X'_j = v'_j\right)$$

Or, La politique π n'est définie que pour le sous-MDP énuméré qui correspond, par définition de Π, à l'instanciation des variables d'état $\Pi(\pi)$. Ainsi, toutes les transitions de π dont l'état de départ et l'état d'arrivée constituent une instanciation des variables de $\mathcal{K}$ différente de $\Pi(\pi)$, sont nulles (inexistantes) :

$$\tilde{T}_p^{\pi}\left(\bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p\right)(\tilde{v}'_p) = 0 \qquad \textbf{si} \qquad \bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j \not\subset \Pi(\pi)$$

En revanche, la transition de π dont l'état de départ est $(\tilde{v}_p, \Pi(\pi))$, représente la macro-probabilité de transiter de l'état abstrait $\tilde{v}_p$ vers l'état abstrait $\tilde{v}'_p$ en suivant la politique π. Cette macro-probabilité est définie dans le sous-MDP abstrait issu de la décomposition du sous-MDP énuméré pour l'instanciation $\Pi(\pi)$, sous l'hypothèse de découplage ($\Pi(\pi)$ est quasiment constant durant l'application de π) :

$$\tilde{T}_p^{\pi}\left(\bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p\right)(\tilde{v}'_p) = \tilde{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}(\tilde{v}_p, \tilde{v}'_p) \qquad \textbf{si} \qquad \bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j = \Pi(\pi)$$

Considérons dès lors une variable d'état X_i quelconque, $i \neq p$. Comme pour $\tilde{T}_p^{\pi}$, la transition $\tilde{T}_i$ est nulle sur tous les états de départ et d'arrivée différents non inclus dans $\Pi(\pi)$:

$$\tilde{T}_i\left(\pi, \bigwedge_{j\neq p} v_j, \bigwedge_{j\neq\{i,p\}} v'_j, \tilde{v}_p\right)(\tilde{v}'_i) = 0 \qquad \textbf{si} \qquad \bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j \not\subset \Pi(\pi)$$

En revanche, si l'état de départ est inclus dans $(\tilde{v}_p, \Pi(\pi))$, la probabilité de la transition $\tilde{T}_i^{\pi}$ peut être calculée à l'aide de la formule de Bayes.
Comme : "$\tilde{X}_p = \tilde{v}_p$" $\Longleftrightarrow \bigcup_{v_p \in \tilde{v}_p}$ "$X_p = v_p$" :

$$\tilde{T}^{\pi}\left(\bigwedge_{j\neq p} v_j, \tilde{v}_p, \bigwedge_{j\neq\{i,p\}} v'_j, \tilde{v}'_p\right)(v'_i)$$

$$= P\left(X'_i = v'_i \mid \pi, \bigwedge_{j\neq p} X_j = v_j, \tilde{X}_p = \tilde{v}_p, \bigwedge_{j\neq\{i,p\}} X'_j = v'_j, \tilde{X}'_p = \tilde{v}'_p\right)$$

$$= \frac{\sum_{v_p\in\tilde{v}_p} P\left(X'_i = v'_i \mid \pi, \bigwedge_{j\neq p} X_j = v_j, X_p = v_p, \bigwedge_{j\neq\{i,p\}} X'_j = v'_j, \tilde{X}'_p = \tilde{v}'_p\right)\cdot P\left(X_p = v_p \mid \pi\right)}{\sum_{v_p\in\tilde{v}_p} P\left(X_p = v_p \mid \pi\right)}$$

car X_p, variable pré-action, ne dépend ni des autres variables pré-action, ni des variables post-action. De plus, la probabilité que $X_p = v_p$, sachant que la politique π est appliquée, est $\frac{1}{|\tilde{v}_p|}$ puisque π n'est applicable que sur la partition $\tilde{v}_p$ et que tous les états de la région $\tilde{v}_p$ sont des états de départ équiprobables de la politique π. Donc :

$$\tilde{T}_i^{\pi}\left(\bigwedge_{j\neq p} v_j, \tilde{v}_p, \bigwedge_{j\neq\{i,p\}} v'_j, \tilde{v}'_p\right)(v'_i)$$

$$= \frac{1}{|\tilde{v}_p|}\sum_{v_p\in\tilde{v}_p} P\left(X'_i = v'_i \mid \pi, \bigwedge_{j\neq p} X_j = v_j, X_p = v_p, \bigwedge_{j\neq\{i,p\}} X'_j = v'_j, \tilde{X}'_p = \tilde{v}'_p\right)$$

$$= \frac{1}{|\tilde{v}_p|}\sum_{v_p\in\tilde{v}_p} \frac{P\left(X'_i = v'_i, \tilde{X}'_p = \tilde{v}'_p \mid \pi, \bigwedge_{j\neq p} X_j = v_j, X_p = v_p, \bigwedge_{j\neq\{i,p\}} X'_j = v'_j\right)}{P\left(\tilde{X}'_p = \tilde{v}'_p \mid \pi, \bigwedge_{j\neq p} X_j = v_j, X_p = v_p, \bigwedge_{j\neq\{i,p\}} X'_j = v'_j\right)}$$

Les réseaux bayésiens dynamiques étant acycliques, soit X'_p ne dépend pas de X'_i, soit X'_i ne dépend pas de X'_p. L'événement "$(X'_i = v'_i, X'_p = v'_p)$" est donc égal soit à l'événement "$(X'_i = v'_i \mid X'_p = v'_p)$", soit à l'événement "$(X'_p = v'_p \mid X'_i = v'_i)$". Appliquons la loi de Bayes à l'égalité précédente, en utilisant l'équivalence d'événements $(\tilde{X}'_p = \tilde{v}'_p) \Leftrightarrow \cup_{v'_p\in\tilde{v}'_p}(X'_p = v'_p)$:

– X'_p ne dépend pas de X'_i

$$\tilde{T}_i^{\pi}\left(\bigwedge_{j\neq p} v_j, \tilde{v}_p, \bigwedge_{j\neq\{i,p\}} v'_j, \tilde{v}'_p\right)(v'_i) =$$

$$\frac{1}{|\tilde{v}_p|}\sum_{v_p\in\tilde{v}_p}\frac{\sum_{v'_p\in\tilde{v}'_p} P\left(X'_i=v'_i \mid \pi, \bigwedge_{j\neq p} X_j=v_j, X_p=v_p, \bigwedge_{j\neq\{i,p\}} X'_j=v'_j, X'_p=v'_p\right)\cdot P\left(X'_p=v'_p \mid \pi, \bigwedge_{j\neq p} X_j=v_j, X_p=v_p, \bigwedge_{j\neq\{i,p\}} X'_j=v'_j\left[, X'_i=v'_i\right]\right)}{\sum_{v'_p\in\tilde{v}'_p}\sum_{v'_{i,k}\in V_i} P\left(X'_i=x'_{i,k} \mid \pi, \bigwedge_{j\neq p} X_j=v_j, X_p=v_p, \bigwedge_{j\neq\{i,p\}} X'_j=v'_j, X'_p=v'_p\right)\cdot P\left(X'_p=v'_p \mid \pi, \bigwedge_{j\neq p} X_j=v_j, X_p=v_p, \bigwedge_{j\neq\{i,p\}} X'_j=v'_j\left[, X'_i=v'_i\right]\right)}$$

$$= \frac{1}{|\tilde{v}_p|}\sum_{v_p\in\tilde{v}_p}\frac{\sum_{v'_p\in\tilde{v}'_p}\mathcal{T}_i^{\pi}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)(v'_i)\cdot\mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(v_p, v'_p\right)}{\sum_{v'_p\in\tilde{v}'_p}\sum_{v'_{i,k}\in V_i} T_i^{\pi}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)\left(v'_{i,k}\right)\cdot\mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(v_p, v'_p\right)}$$

– X'_i ne dépend pas de X'_p

$$\tilde{T}_i^{\pi}\left(\bigwedge_{j\neq p} v_j, \tilde{v}_p, \bigwedge_{j\neq\{i,p\}} v'_j, \tilde{v}'_p\right)(v'_i) =$$

$$\frac{1}{|\tilde{v}_p|}\sum_{v_p\in\tilde{v}_p}\frac{\sum_{v'_p\in\tilde{v}'_p} P\left(X'_p=v'_p \mid \pi, \bigwedge_{j\neq p} X_j=v_j, X_p=v_p, \bigwedge_{j\neq p} X'_j=v'_j\right)\cdot P\left(X'_i \mid \pi, \bigwedge_{j\neq p} X_j=v_j, X_p=v_p, \bigwedge_{j\neq\{i,p\}} X'_j=v'_j\left[, X'_p=v'_p\right]\right)}{\sum_{v'_p\in\tilde{v}'_p}\sum_{v'_{i,k}\in V_i} P\left(X'_p=v'_p \mid \pi, \bigwedge_{j\neq p} X_j=v_j, X_p=v_p, \bigwedge_{j\neq\{i,p\}} X'_j=v'_j, X'_i=v'_{i,k}\right)\cdot P\left(X'_i=v'_{i,k} \mid \pi, \bigwedge_{j\neq p} X_j=v_j, X_p=v_p, \bigwedge_{j\neq\{i,p\}} X'_j=v'_j\left[, X'_p=v'_p\right]\right)}$$

$$= \frac{1}{|\tilde{v}_p|}\sum_{v_p\in\tilde{v}_p}\frac{\sum_{v'_p\in\tilde{v}'_p}\mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(v_p, v'_p\right)\cdot T_i^{\pi}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)(v'_i)}{\sum_{v'_p\in\tilde{v}'_p}\sum_{v'_{i,k}\in V_i}\mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(v_p, v'_p\right)\cdot T_i^{\pi}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)\left(v'_{i,k}\right)}$$

Ainsi, dans tous les cas :

$$\tilde{T}_i^{\pi}\left(\bigwedge_{j\neq p} v_j, \tilde{v}_p, \bigwedge_{j\neq\{i,p\}} v'_j, \tilde{v}'_p\right)(v'_i) =$$

$$\frac{1}{|\tilde{v}_p|}\sum_{v_p\in\tilde{v}_p}\frac{\sum_{v'_p\in\tilde{v}'_p}\mathcal{T}_i^{\pi}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)(v'_i)\cdot\mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(v_p, v'_p\right)}{\sum_{v'_p\in\tilde{v}'_p}\sum_{v'_{i,k}\in V_i} T_i^{\pi}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)\left(v'_{i,k}\right)\cdot\mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(v_p, v'_p\right)}$$

avec :

– T_i^{π} est la probabilité agrégée d'obtenir les différentes valeurs de X_i en suivant la politique π **pour le MDP factorisé initial**

– $T^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}$ est la probabilité agrégée de transiter d'un état de la région où est définie π vers un état périphérique de cette région **pour le MDP énuméré correspondant à l'instanciation** $\left(\wedge_{j\neq p} X_j = v_j, \wedge_{j\neq p} X'_j = v'_j\right)$

La deuxième probabilité agrégée est calculée durant la décomposition des sous-MDP énumérés par rapport à X_p : elle sert à calculer les macro-probabilités de transition $\tilde{T}_{\wedge_{j\neq p} v_j \wedge_{j\neq p} v'_j}$ des sous-MDP énumérés.

Il ne reste plus qu'à calculer la fonction de macro-récompense du MDP factorisé abstrait. $\tilde{R}^\pi\left(\wedge_{j\neq p} v_j, \wedge_{j\neq p} v'_j, \tilde{v}_p, \tilde{v}'_p\right)$ est la moyenne des récompenses accumulées en suivant π, en partant de l'état $(\wedge_{j\neq p} v_j, \tilde{v}_p)$ et en arrivant à l'état $\left(\wedge_{j\neq p} v'_j, \tilde{v}'_p\right)$. Or π n'est applicable, par définition, que sur $\Pi(\pi)$ si bien que cette macro-récompense moyenne est nulle si l'état de départ n'est pas inclus dans $\Pi(\pi)$:

$$\tilde{R}^\pi\left(\bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p, \tilde{v}'_p\right) = 0 \qquad \textbf{si} \qquad \bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j \not\subset \Pi(\pi)$$

Sinon, sous l'hypothèse de découplage, la récompense moyenne précédente est égale à la moyenne des récompenses cumulées en suivant π depuis $\tilde{v}_p$ vers $\tilde{v}'_p$ **en supposant que** $\Pi(\pi)$ **est constant**. Dans ce cas, la récompense moyenne est égale à la macro-récompense de la transition $\left(\pi, \tilde{v}_p, \tilde{v}'_p\right)$ pour le sous-MDP énuméré correspondant à l'instanciation $\left(\wedge_{j\neq p} v_j, \wedge_{j\neq p} v'_j\right)$:

$$\tilde{R}^\pi\left(\bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p, \tilde{v}'_p\right) = \tilde{R}^\pi_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(\tilde{v}_p, \tilde{v}'_p\right) \qquad \textbf{si} \qquad \bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j \subset \Pi(\pi)$$

□

Dans l'expression des probabilités de transition du MDP abstrait factorisé (théorème 9) :

- T^π_i est la probabilité agrégée d'obtenir les différentes valeurs de X_i en suivant la politique π **pour le MDP factorisé initial**
- $T^\pi_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}$ est la probabilité agrégée de transiter d'un état de la région où est définie π vers un état périphérique de cette région **pour le MDP énuméré correspondant à l'instanciation** $\left(\wedge_{j\neq p} X_j = v_j, \wedge_{j\neq p} X'_j = v'_j\right)$

Alors que la deuxième probabilité a déjà été calculée durant la décomposition des sous-MDP énumérés afin de calculer les macro-transitions des sous-MDP énumérés, la première probabilité doit être calculée. La probabilité agrégrée d'obtenir les différentes valeurs de X_i en suivant la politique π est donnée dans le théorème 10.

Démonstration. Précisons d'abord quels sont les instants pré-action et post-action. L'instant pré-action est celui où commence l'application de la politique π, que nous noterons π^0. L'instant post-action est celui où finit l'application de π. Cet instant est théoriquement infini à cause des cycles éventuels dans les transitions entre les états énumérés d'une région d'un sous-MDP énumérés. Nous le noterons donc π^∞. En pratique, la politique π s'arrête soit lorsqu'une variable de $\Pi(\pi)$ change, ce qui

Théorème 10 *Probabilités agrégées*

Soient un MDP factorisé $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, X_p une variable de grande arité, et π une politique locale par rapport à X_p définie sur $\Pi(\pi)$.
Si la chaîne de Markov définie par les transitions entre les états énumérés du MDP factorisé initial pour la politique π est **ergodique**, *alors les probabilités agrégées d'obtenir les différentes valeurs des variables d'état X_i en suivant π, sachant $\left(\wedge_{1 \leqslant j \leqslant n} X_j = v_j, \wedge_{j \neq i} X'_j = v'_j\right)$ avec $(v_1, \ldots, v_n) \subset \Pi(\pi)$, sont solutions du système (non linéaire) suivant :*

$$\forall\, 1 \leqslant i \leqslant n\ ,\ T_i^{\pi}\left(\bigwedge_{1 \leqslant j \leqslant n} v_j, \bigwedge_{j \neq i} v'_j\right)(v'_i) =$$

$$\sum_{\left(v''_1, \ldots, v''_n\right) \in \Pi(\pi) \cup V_p} T_i^{\pi\left(v''_p\right)}\left(\bigwedge_{1 \leqslant j \leqslant n} v''_j, \bigwedge_{j \neq i} v'_j\right)(v'_i) \prod_{k=1}^{n} T_k^{\pi}\left(\bigwedge_{1 \leqslant j \leqslant n} v_j, \bigwedge_{j \neq k} v''_j\right)(v''_k)$$

Cette solution peut être calculée itérativement sur le système précédent en partant de la distribution déterministe sur $(v_1, \ldots, v_n)$.

rend la politique inapplicable puisqu'elle n'est définie que sur $\Pi(\pi)$, soit lorsque l'agent sort de la région où est définie la politique. Néanmoins, **la probabilité que la politique π s'arrête au bout d'un temps fini est non nulle**, à moins que la politique soit uniquement ré-entrante et qu'elle ne change jamais les valeurs des variables d'état de $\Pi(\pi)$.

D'après ce que nous venons de mentionner, pour tout $1 \leqslant j \leqslant n$, $X_j = X_j^{\pi^0}$ et $X'_j = X_j^{\pi^\infty}$. Soit π^t un instant du processus d'application de la politique π. Calculons, pour une variable X_i, la probabilité que "$X_i^{\pi^{t+1}} = v'_i$" sachant "$\left(\wedge_{1 \leqslant j \leqslant n} X_j^{\pi^0}\right) = v_j$" et "$\wedge_{j \neq i} X_j^{\pi^{t+1}} = v'_j$" en suivant la politique π. Comme la politique π n'est définie que sur $\Pi(\pi)$, il est nécessaire que $(v_1, \ldots, v_n) \subset \Pi(\pi)$. L'événement "$X_i^{\pi^{t+1}} = v'_i$" peut être réalisé en venant de différents états à l'instant π^t :

$$P\left(X_i^{\pi^{t+1}} = v'_i \mid \pi, \bigwedge_{1 \leqslant j \leqslant n} X_j^{\pi^0} = v_j, \bigwedge_{j \neq i} X_j^{\pi^{t+1}} = v'_j\right) =$$

$$\sum_{e^{\pi^t} \in \mathcal{V}} P\left(\left(X_1^{\pi^t}, \ldots, X_n^{\pi^t}\right) = e^{\pi^t} \mid \pi, \bigwedge_{1 \leqslant j \leqslant n} X_j^{\pi^0} = v_j\right) \cdot$$

$$P\left(X_i^{\pi^{t+1}} = v'_i \mid \pi\left(e^{\pi^t}\right), \left(X_1^{\pi^t}, \ldots, X_n^{\pi^t}\right) = e^{\pi^t}, \bigwedge_{j \neq i} X_j^{\pi^{t+1}} = v'_j\right) =$$

$$\sum_{\left(v_1'',\ldots,v_n''\right)\in\mathcal{V}} P\left(\bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^t}=v_j'' \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j\right)\cdot$$

$$P\left(X_i^{\pi^{t+1}}=v_i' \mid \pi\left(\bigwedge_{1\leqslant j\leqslant n} v_j''\right), \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^t}=v_j'', \bigwedge_{j\neq i} X_j^{\pi^{t+1}}=v_j'\right)=$$

$$\sum_{\left(v_1'',\ldots,v_n''\right)\in\Pi(\pi)\cup V_p}\prod_{k=1}^{n} P\left(X_k^{\pi^t}=v_k'' \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j, \bigwedge_{j\neq k} X_j^{\pi^t}=v_j''\right)\cdot$$

$$T_i\left(\pi\left(v_p''\right), \bigwedge_{1\leqslant j\leqslant n} v_j'', \bigwedge_{j\neq i} v_j'\right)\left(v_i'\right)$$

En effet, $\pi\left(\wedge_{1\leqslant j\leqslant n} v_j''\right)=\pi\left(v_p''\right)$ car π est une politique locale n'agissant que sur la variable X_p. De plus, π n'étant définie que sur $\Pi(\pi)$, la somme précédente est nulle sur toutes les instanciations de variables n'appartenant pas à $\Pi(\pi)$.
Soit $(u_t)_{t\in\mathbb{N}}$ une suite de $\mathbb{R}^N$, $N=n\prod_{j=1}^{n}|V_j|$, définie par :

$$\forall\, t\in\mathbb{N}\ ,\ u_t=\left[\begin{array}{c} P\left(X_1^{\pi^t}=v_1' \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j, \bigwedge_{j\neq 1} X_j^{\pi^t}=v_j'\right) \\ \vdots \\ P\left(X_n^{\pi^t}=v_n' \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j, \bigwedge_{j\neq n} X_j^{\pi^t}=v_j'\right) \end{array}\right]_{\left(v_1',\ldots,v_n'\right)\in\mathcal{V}}$$

Soit f la fonction de $\mathbb{R}^N$ dans $\mathbb{R}^N$ définie par :

$$\forall\ \left(v_1',\ldots,v_n'\right)\in\mathcal{V}\ ,\ \forall\ 1\leqslant i\leqslant n\ ,\ f_{\left(v_1',\ldots,v_n'\right),i}\left(\left[\begin{array}{c} x_{v'',1} \\ \vdots \\ x_{v'',n} \end{array}\right]_{v''\in\mathcal{V}}\right)=$$

$$\sum_{\left(v_1'',\ldots,v_n''\right)\in\Pi(\pi)\cup V_p} T_i^{\pi\left(v_p''\right)}\left(\bigwedge_{1\leqslant j\leqslant n} v_j'', \bigwedge_{j\neq i} v_j'\right)\left(v_i'\right)\prod_{k=1}^{n} x_{\left(v_1'',\ldots,v_n''\right),k}$$

La suite $(u_t)_{t\in\mathbb{N}}$ vérifie :

$$\forall\, t\in\mathbb{N}\ ,\ u_{t+1}=f\left(u_t\right) \tag{3.1}$$

Soit la fonction g de $\mathbb{R}^N$ dans $\mathbb{R}^{\tilde{N}}$, $\tilde{N} = \dfrac{N}{n}$, définie par :

$$\forall\ (v'_1, \ldots, v'_n) \in \mathcal{V}\ ,\ g_{\left(v'_1,\ldots,v'_n\right)}\left(\left[\begin{array}{c} x_{i,1} \\ \vdots \\ x_{i,n} \end{array}\right]_{i\in\mathcal{V}}\right) = \prod_{k=1}^{n} x_{\left(v'_1,\ldots,v'_n\right),k}$$

Soit la suite $(w_t)_{t\in\mathbb{N}}$ de $\mathbb{R}^{\tilde{N}}$ définie par : $\forall\ t \in \mathbb{N}\ ,\ w_t = g\left(u_t\right)$
La suite $(w_t)_{t\in\mathbb{N}}$ correspond à la probabilité conjointe des variables d'état à l'instant t. Il sera alors possible de démontrer la convergence de cette suite à l'aide du théorème ergodique de Chacon-Ornstein [45] (existence d'une unique probabilité stationnaire pour une chaîne de Markov ergodique).
Soit donc la fonction h de $\mathbb{R}^{\tilde{N}}$ dans $\mathbb{R}^{\tilde{N}}$ représentant la matrice de transition des probabilités jointes. Cette fonction est définie par :

$$\forall\ (v'_1, \ldots, v'_n) \in \mathcal{V}\ ,\ h_{\left(v'_1,\ldots,v'_n\right)}\left(\left(y_{v''}\right)_{v''\in\mathcal{V}}\right) = \sum_{\left(v''_1,\ldots,v''_n\right)\in\Pi(\pi)\cup V_p} y_{\left(v''_1,\ldots,v''_n\right)} \prod_{i=1}^{n} T_i^{\pi\left(v''_p\right)}\left(\bigwedge_{1\leqslant j\leqslant n} v''_j, \bigwedge_{j\neq i} v'_j\right)\left(v'_i\right)$$

Montrons que l'équation récursive 3.1 implique : $\forall\ t \in \mathbb{N}\ ,\ w_{t+1} = h\left(w_t\right)$, afin de prouver la convergence de la suite $(w_t)_{t\in\mathbb{N}}$.

$$\forall\ t \in \mathbb{N}\ ,\ w_{t+1} = g\left(u_{t+1}\right)$$

Donc :

$$w_{t+1}\left(v'_1, \ldots, v'_n\right) = g_{\left(v'_1,\ldots,v'_n\right)}\left(u_{t+1}\right) = \prod_{i=1}^{n} u_{t+1}\left(v'_1, \ldots, v'_n\right)(i) =$$
$$\prod_{i=1}^{n} P\left(X_i^{\pi^{t+1}} = v'_i \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^0 = v_j, \bigwedge_{j\neq i} X_j^{\pi^{t+1}} = v'_j\right) =$$
$$P\left(\bigwedge_{1\leqslant i\leqslant n} X_i^{\pi^{t+1}} = v'_i \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^0 = v_j\right) =$$

$$\sum_{\left(v''_1,\ldots,v''_n\right)\in\Pi(\pi)\cup V_p} P\left(\bigwedge_{1\leqslant i\leqslant n} X_i^{\pi^{t+1}}=v'_i \mid \pi\left(v''_p\right), \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^t}=v''_j\right)\cdot$$

$$P\left(\bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^t}=v''_j \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^0=v_j\right)=$$

$$\sum_{\left(v''_1,\ldots,v''_n\right)\in\Pi(\pi)\cup V_p}\prod_{i=1}^{n} P\left(X_i^{\pi^{t+1}}=v'_i \mid \pi\left(v''_p\right), \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^t}=v''_j, \bigwedge_{j\neq i} X_j^{\pi^{t+1}}=v'_j\right)\cdot$$

$$\prod_{k=1}^{n} P\left(X_k^{\pi^t}=v''_k \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^0=v_j, \bigwedge_{j\neq k} X_j^{\pi^t}=v''_j\right)=$$

$$\sum_{\left(v''_1,\ldots,v''_n\right)\in\Pi(\pi)\cup V_p}\prod_{i=1}^{n} T_i^{\pi\left(v''_p\right)}\left(\bigwedge_{1\leqslant j\leqslant n} v''_j, \bigwedge_{j\neq i} v'_j\right)\left(v'_i\right)\cdot w_t\left(v'_1,\ldots,v'_n\right)=h_{\left(v'_1,\ldots,v'_n\right)}\left(w_t\right)$$

Donc : $\forall\, t\in\mathbb{N}\,,\ w_{t+1}=h\left(w_t\right)$
Or h représente l'opérateur de mise à jour des probabilités conjointes, c'est-à-dire des probabilités de la chaîne de Markov correspondant à la politique π et dont les états sont les états *énumérés* du MDP. De plus, cette chaîne de Markov est par hypothèse ergodique. Ainsi, d'après le théorème ergodique de Chacon-Ornstein [45], la suite $(w_t)_{t\in\mathbb{N}}$ converge vers une unique distribution stationnaire de probabilité jointe w_∞.
Or, pour chaque instant t, u_t est une somme finie de probabilités jointes w_t :

$$\forall\, t\in\mathbb{N}\,,\ u_t\left(v'_1,\ldots,v'_n\right)(i)=P\left(X_i^{\pi^t}=v'_i \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j, \bigwedge_{j\neq i} X_j^{\pi^t}=v'_j\right)=$$

$$\frac{P\left(\bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^t}=v'_j \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j\right)}{P\left(\bigwedge_{j\neq i} X_j^{\pi^t}=v'_j \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j\right)}=$$

$$\frac{\prod_{k=1}^{n} P\left(X_k^{\pi^t}=v'_k \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j, \bigwedge_{j\neq k} X_j^{\pi^t}=v'_j\right)}{\sum_{v''_i\in V_i} P\left(\bigwedge\left(X_i^{\pi^t}=v''_i\right)\bigwedge_{j\neq i} X_j^{\pi^t}=v'_j \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0}=v_j\right)}=$$

$$\frac{w_t\left(v'_1,\ldots,v'_n\right)}{\sum_{v''_i\in V_i} w_t\left(v'_1,\ldots,v'_{i-1},v''_i,v'_{i+1},\ldots,v'_n\right)}$$

Donc : $\forall\, t \in \mathbb{N}\ ,\ u_t = q\,(w_t)$ où q est la fonction de $\mathbb{R}^{\tilde{N}}$ dans $\mathbb{R}^N$ définie par :

$$\forall\ (v'_1,\ldots,v'_n) \in \mathcal{V}\ ,\ \forall\ 1 \leqslant i \leqslant n\ ,\ q_{\left(v'_1,\ldots,v'_n\right),i}\left((x_{v''})_{v''\in\mathcal{V}}\right) = \frac{x_{\left(v'_1,\ldots,v'_n\right)}}{\sum\limits_{v''_i\in V_i} x_{\left(v'_1,\ldots,v'_{i-1},v''_i,v'_{i+1},\ldots,v'_n\right)}}$$

Comme q est continue et que la suite $(w_t)_{t\in\mathbb{N}}$ converge, on en déduit que la suite $(u_t)_{t\in\mathbb{N}}$ converge (vers $u_\infty = q(w_\infty)$). Il existe donc une probabilité stationnaire des variables d'état en suivant la politique π et en partant de la distribution $\left(X_1^{\pi^0},\ldots,X_n^{\pi^0}\right) = (v_1,\ldots,v_n)$.
Comme f est également continue est que la suite $(u_t)_{t\in\mathbb{N}}$ converge, on en déduit en passant à la limite dans l'équation 3.1 que :

$$u_\infty = f\,(u_\infty)$$

qui est l'équation donnée dans l'énoncé du théorème en notant :

$$u_\infty = \left[\begin{array}{c} T_1^\pi\left(\bigwedge\limits_{1\leqslant j\leqslant n} v_j, \bigwedge\limits_{j\neq 1} v'_j\right)(v'_1) \\ \vdots \\ T_n^\pi\left(\bigwedge\limits_{1\leqslant j\leqslant n} v_j, \bigwedge\limits_{j\neq n} v'_j\right)(v'_n) \end{array}\right]_{\left(v'_1,\ldots,v'_n\right)\in\mathcal{V}}$$

Il ne reste plus qu'à préciser la distribution initiale car le calcul du point fixe précédent dépend au premier élément u_0 (contrairement à l'équation de Bellman). Calculons pour cela la première itération du processus :

$$\forall\ 1 \leqslant i \leqslant n\ ,\ P\left(X_i^{\pi^1} = v'_i \mid \pi, \bigwedge_{1\leqslant j\leqslant n} X_j^{\pi^0} = v_j, \bigwedge_{j\neq i} X_j^{\pi^1} = v'_j\right) =$$
$$T_i^{\pi(v_p)}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)(v'_i) =$$
$$\sum_{\left(v''_1,\ldots,v''_n\right)\in\Pi(\pi)\cup V_p} T_i^{\pi\left(v''_p\right)}\left(\bigwedge_{1\leqslant j\leqslant n} v''_j, \bigwedge_{j\neq i} v'_j\right)(v'_i) \prod_{k=1}^{n} u_0\,(v''_1,\ldots,v''_n)\,(k)$$

avec : $\forall\ (v''_1,\ldots,v''_n) \in \mathcal{V}\ ,\ \forall\ 1 \leqslant k \leqslant n\ ,\ u_0\,(v''_1,\ldots,v''_n)\,(k) = \left\{\begin{array}{l} 1 \text{ si } v'' = v \\ 0 \text{ sinon} \end{array}\right.$ □

Ainsi, nous disposons de deux théorèmes qui montrent qu'il est possible de réduire l'espace d'états d'un MDP factorisé, en appliquant des techniques de décomposition de MDP énumérés sur les sous-espaces engendrés par des variables d'état de

grande arité. De plus, ces théorèmes fournissent les macro-transitions et les macro-récompenses du MDP factorisé réduit. Les macro-transitions peuvent être calculées itérativement grâce à des équations de programmation dynamique portant sur les arbres de transition des variables d'état.

Nous pouvons donc en déduire un algorithme de réduction automatique de MDP factorisés, qui fera l'objet du chapitre suivant. Cet algorithme est la généralisation des algorithmes de décomposition de MDP énumérés aux MDP factorisés. Le principe de réduction automatique d'arité des variables d'état ayant été étudié récemment en complément de nos autres contributions, nous n'avons pas eu le temps d'implémenté cet algorithme. Aussi, nous discuterons de l'efficacité de notre approche en calculant les complexités des différentes routines utilisées. Seule une mise en œuvre permettrait toutefois d'évaluer correctement notre approche.

Chapitre 4

Algorithme de réduction automatique d'arité des variables d'état dans les MDP factorisés

L'étude théorique nous a permis de développer l'algorithme 5 de réduction automatique de MDP factorisés, appelé `SVAR` («State Variable Arity Reduction»). Cet algorithme appelle, dans l'ordre :

- `CalculeDependance` : calcule $\mathcal{K}_p$, c'est-à-dire les variables d'état dont dépend X_p et les arbres de récompense pour toutes les actions (cf. algorithme 6)
- `EnumereSousMDP` : énumère les sous-MDP par rapport à X_p pour chaque instanciation de $\mathcal{K}_p$ (cf. algorithme 7)
- `PartitionneSousMDP` : calcule le partitionnement optimal commun à tous les graphes de transition des sous-MDP énumérés par rapport à X_p (cf. algorithme 8)
- `DecomposeSousMDP` : Calcule les politiques locales pour chaque région du partitionnement $\tilde{V}_p$, et chaque MDP sous-énuméré par rapport à X_p (c'est-à-dire chaque instanciation de $\mathcal{K}_p$) (cf. algorithme 9)
- `GenereMDPabstrait` : Génère le MDP abstrait factorisé dont la variable X_p est réduite à $\tilde{X}_p$ et les actions sont les politiques locales précédemment calculées (cf. algorithme 10)

4.1 Fonction `CalculeDependance`

Cette fonction calcule l'ensemble des variables d'état, privé de X_p et X'_p dont dépendent X_p et la fonction de récompense $\mathcal{R}$ pour toutes les actions du MDP factorisé initial. L'ensemble $\mathcal{K}_p$ est obtenu en parcourant, pour chaque action a, l'arbre de transition T_p^a de la variable X_p et l'arbre de récompense R^a de manière

Algorithme 5 : Fonction `SVAR`

Données : $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ (MDP factorisé), p
Résultat : $\langle \tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$ (MDP factorisé)
début
  $\mathcal{K}_p \leftarrow$ `CalculeDependance` $(X_p, \mathcal{R})$;
  pour $e \in \mathcal{K}_p$ **faire**
    $\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle \leftarrow$ `EnumereSousMDP` (X_p, e);
  $\tilde{V}_p \leftarrow$ `PartitionneSousMDP` $\left(\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle_{e \in \mathcal{K}_p}\right)$;
  pour $e \in \mathcal{K}_p$ **faire**
    $\langle \tilde{V}_p, \tilde{\mathcal{A}}_e, \tilde{\mathcal{T}}_e, \tilde{\mathcal{R}}_e \rangle \leftarrow$ `DecomposeSousMDP` $\left(\tilde{V}_p, \langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle\right)$;
  $\langle \tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle \leftarrow$ `GenereMDPabstrait` $\left(\langle \tilde{V}_p, \tilde{\mathcal{A}}_e, \tilde{\mathcal{T}}_e, \tilde{\mathcal{R}}_e \rangle_{e \in \mathcal{K}_p}\right)$;
  retourner $\langle \tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$;
fin

exhaustive. Le parcours peut être récursif ou itératif ; l'algorithme 6 présente une version itérative de la fonction `CalculeDependance`.

A titre d'exemple, considérons le MDP factorisé défini dans la figure 3.6. La fonction `CalculeDependance` commence, en parcourant l'arbre de transition de X, par ajouter Z et Y' dans $\mathcal{K}_X$. Le parcours de l'arbre de récompense retourne ensuite la variable Z, qui est déjà incluse dans $\mathcal{K}_p$.

Calculons la complexité de cet algorithme en fonction de d_p, le nombre maximum de variables dont dépend X_p pour toutes les actions, et d_r, le nombre de variables dont dépend $\mathcal{R}$ pour toutes les actions. Soit d la longueur d'une instanciation de $\mathcal{K}_p$: $d_p \leqslant d$ et $d_r \leqslant d$.
Soit C le cardinal du plus grand ensemble de variables d'état : $C = \max_{1 \leqslant i \leqslant n} |V_i|$.
Si on appelle $\left(V_{i_1}, \cdots, V_{i_{d_p}}\right)$ la liste des variables dont dépend l'arbre de probabilité de X_p pour l'action a, la complexité de la recherche dans l'arbre de probabilité de X_p est :

$$\begin{aligned} 5 + 2|V_{i_1}| + |V_{i_1}|\left(5 + 2|V_{i_2}| + |V_{i_2}|\left(\cdots\right)\right) &= 5 + 7\sum_{k=1}^{d_p-1}\prod_{j=1}^{k}|V_{i_j}| + \prod_{j=1}^{d_p}|V_{i_j}| \\ &\leqslant 5 + 7\sum_{k=1}^{d_p-1} C^k + C^{d_p} \\ &= 5 + 7\frac{1 - C^{d_p-1}}{1 - C} + C^{d_p} \end{aligned}$$

Algorithme 6 : Fonction `CalculeDependance` (itératif, ordre en profondeur)

Données : $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ (MDP factorisé), p
Résultat : $\mathcal{K}_p = \mathcal{D}(X_p) \setminus \{X_p\} \times \mathcal{D}(\mathcal{R}) \setminus \{X_p, X'_p\}$
début
  $\mathcal{K}_p \leftarrow \emptyset$;
  pour $a \in \mathcal{A}$ **faire**
    $NoeudsVisites = \emptyset$;
    si $T_p^a.racine().type() = noeud$ **alors**
      $NoeudsVisites.Enfiler(T_p^a.racine())$;
      tant que $NoeudsVisites \neq \emptyset$ **faire**
        $noeud \leftarrow NoeudsVisites.Defiler()$;
        si $fils \neq X_p$ **et** $fils \notin \mathcal{K}_p$ **alors**
          $\mathcal{K}_p \leftarrow fils$;
        pour $fils \in noeud.fils()$ **faire**
          si $fils.type() = noeud$ **alors**
            $NoeudsVisites.Enfiler(fils)$;
    $NoeudsVisites = \emptyset$;
    si $R^a.racine().type() = noeud$ **alors**
      $NoeudsVisites.Enfiler(R^a.racine())$;
      tant que $NoeudsVisites \neq \emptyset$ **faire**
        $noeud \leftarrow NoeudsVisites.Defiler()$;
        si $fils \neq X_p$ **et** $fils \neq X'_p$ **et** $fils \notin \mathcal{K}_p$ **alors**
          $\mathcal{K}_p \leftarrow fils$;
        pour $fils \in noeud.fils()$ **faire**
          si $fils.type() = noeud$ **alors**
            $NoeudsVisites.Enfiler(fils)$;
  retourner $\mathcal{K}_p$;
fin

De même, en appelant $\left(V_{j_1}, \cdots, V_{j_{d_r}}\right)$ la liste des variables dont dépend l'arbre de récompense, la complexité de la recherche dans l'arbre de récompense est :

$$\begin{aligned} 6 + 8|V_{j_1}| + |V_{j_1}|\left(6 + 2|V_{j_2}| + |V_{j_2}|\left(\cdots\right)\right) &= 6 + 8\sum_{k=1}^{d_r-1}\prod_{i=1}^{k}|V_{j_i}| + \prod_{i=1}^{d_r}|V_{j_i}| \\ &\leqslant 6 + 8\sum_{k=1}^{d_r-1} C^k + C^{d_r} \\ &= 6 + 8\frac{1 - C^{d_r-1}}{1 - C} + C^{d_r} \end{aligned}$$

La complexité de la fonction `CalculeDependance` est donc majorée par :

$$\begin{aligned} 1 + |\mathcal{A}|\left(3 + 5 + 7\frac{1 - C^{d_p-1}}{1 - C} + C^{d_p} + 3 + 6 + 8\frac{1 - C^{d_r-1}}{1 - C} + C^{d_r}\right) & \\ \leqslant O\left(|\mathcal{A}|\left(17 + 15\frac{1 - C^{d-1}}{1 - C} + 2C^d\right)\right) & \end{aligned}$$

car $d_p \leqslant d$, $d_r \leqslant d$ et $C > 1$.

Dans le pire cas, $d = 2n - 2$ mais ce cas est très rare puisqu'une variable d'état dépend en général de peu de variables parmi toutes les variables d'un problème de décision. Si n est grand, la complexité est alors $O\left(2|\mathcal{A}|C^{2(n-1)}\right)$.

Dans le cas d'un problème de type RESSAC, la variable de position ne dépend que de l'autonomie et du vent, mais la récompense dépend des variables objectifs. Dans ce cas, $\mathcal{K}_p = V_A \cup V_V \cup_{O \in Objectifs} V_O$ donc $d = n_o + 2$ où n_o est le nombre d'objectifs à atteindre. Si n_o est grand, la complexité est alors $O\left(|\mathcal{A}|C^{n_o+2}\right)$.

4.2 Fonction `EnumereSousMDP`

Etant donnée une instanciation des variables d'état $e \in \mathcal{K}_p$, cette fonction génère le sous-MDP énuméré par rapport à X_p. Pour cela, nous parcourons l'arbre de transition de X_p et l'arbre de récompense $\mathcal{R}$ pour chaque action **en suivant le chemin imposé par l'instanciation** e. Dans les deux cas, nous obtenons un sous-arbre qui peut dépendre de X_p ou X'_p :

- le sous-arbre de transition de X_p contient au plus le nœud X_p ; chaque feuille est une liste de probabilités de transition vers chaque valeur de X'_p (sachant les différentes valeurs de départ X_p si le sous-arbre contient la variable X_p)
- le sous-arbre de récompense contient au plus les nœuds X_p et X'_p ; chaque feuille est une récompense (qui dépend de X_p ou X'_p si le sous-arbre contient les variables X_p ou X'_p)

Par exemple, pour le MDP factorisé défini dans la figure 3.6, l'ensemble $\mathcal{K}_X$ est $\{Z, Y'\}$. Le sous-arbre de transition de la variable X pour une instanciation donnée de $\mathcal{K}_X$ contient la variable X. Quant au sous-arbre de récompense, il contient les variables X et X'.

La fonction `EnumereSousMDP` (cf. algorithme 7) parcourt, pour chaque action et chaque valeur de X_p et X'_p, l'arbre de probabilité de X'_p et l'arbre de récompense (les autres variables étant fixées à $e \in \mathcal{K}_p$). L'instanciation e est une liste de valeurs pour chaque variable d'état différente de X_p et X'_p, puisque $\mathcal{K}_p$ ne contient ni X_p ni X'_p.

Une transition entre deux états de V_p n'est ajoutée à l'ensemble des transitions du sous-MDP énuméré que si sa probabilité est non nulle (variable booléenne $TransitionExiste$).

D'autre part, les feuilles de l'arbre de probabilité de X_p sont des listes de probabilités pour chaque valeur de X'_p. La fonction $noeud.feuille(v'_p)$, si $noeud$ est une feuille, renvoie donc l'élément v'_p de la feuille $noeud$ (c'est-à-dire la probabilité que $X'_p = v'_p$).

A titre d'exemple, les sous-MDP énumérés générés par la fonction `EnumereSousMDP`, pour le MDP factorisé défini dans la figure 3.6, sont représentés dans la figure 3.7.

La complexité de la fonction `EnumereSousMDP` est, en notant d la longueur d'une instanciation de $\mathcal{K}_p$ (nombre de variables dont dépend X_p et $\mathcal{R}$, autres que X_p et X'_p) :

$$O\left(2 + |\mathcal{A}||V_p|^2 \left(3 + \max\left(2(d+1), 3\right) + 2 + \max\left(2(d+2), 3\right)\right)\right)$$

Mentionnons 2 cas particuliers :

- pire cas : $d = 2n - 2$, n grand
  Complexité : $O\left(8n|\mathcal{A}||V_p|^2\right)$
- mission de type ReSSAC : $d = n_o{+}2$, n_o grand, où n_o est le nombre d'objectifs du problème de décision (X'_p dépend du vent et de l'autonomie, et $\mathcal{R}$ dépend des variables objectif)
  Complexité : $O\left(4n_o|\mathcal{A}||V_p|^2\right)$

4.3 Fonction `PartitionneSousMDP`

Cette fonction (cf. algorithme 8) construit le MDP sous-jacent de la variable X_p à partir des sous-MDP énumérés par rapport à X_p. Rappelons que le MDP sous-jacent est la réunion de tous les sous-MDP énumérés afin que le partitionnement de l'espace d'états prenne en compte simultanément les transitions de tous les sous-MDP énumérés. La fonction `PartitionneSousMDP` partitionne ensuite le MDP sous-jacent automatiquement en utilisant l'algorithme de [25], correspondant à la fonction `Partitionne`.

Algorithme 7 : Fonction `EnumereSousMDP`

Données : $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ (MDP factorisé), p, e (liste d'instanciation des variables de $\mathcal{K}_p$)
Résultat : $\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle$ (MDP énuméré)
début
  $\mathcal{T}_e \leftarrow \emptyset$, $\mathcal{R}_e \leftarrow \emptyset$;
  pour $a \in \mathcal{A}$ **faire**
    pour $v_p \in V_p$ **faire**
      pour $v'_p \in V_p$ **faire**
        $TransitionExiste \leftarrow$ FALSE;
        $noeud \leftarrow T_p^a.racine()$;
        si $noeud.type() = variable$ **alors**
          pour $v \in (e \cup \{v_p\})$ **faire**
            si $v.variable() = noeud$ **alors**
              $noeud \leftarrow noeud.fils(v.valeur())$;
        sinon
          si $noeud.feuille(v'_p) \neq 0$ **alors**
            $TransitionExiste \leftarrow$ TRUE;
            $\mathcal{T}_e \leftarrow \mathcal{T}_e \cup probabilite\left(a, v_p, v'_p, noeud.feuille(v'_p)\right)$;
        $noeud \leftarrow R^a.racine()$;
        si $noeud.type() = variable$ **alors**
          pour $v \in (e \cup \{v_p, v'_p\})$ **faire**
            si $v.variable() = noeud$ **alors**
              $noeud \leftarrow noeud.fils(v.valeur())$;
        sinon
          si $noeud.feuille() \neq 0$ **et** $TransitionExiste = TRUE$ **alors**
            $\mathcal{R}_e \leftarrow \mathcal{R}_e \cup recompense\left(a, v_p, v'_p, noeud.feuille()\right)$;
  retourner $\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle$;
fin

Cette fonction nécessite en entrée le nombre de macro-états du partitionnement à réaliser (ou le nombre d'états agrégés dans chaque macro-état, le partitionnement étant équilibré). Ce nombre de macro-états, soit $|\tilde{V}_p|$, est renvoyé par la fonction `CalculeDimensionPartitionnement`. Il doit être calculé de manière à minimiser la complexité totale de notre algorithme de réduction automatique d'arité de variables d'état, combinée à celle de l'optimisation du MDP abstrait factorisé. En effet, l'optimisation de ce MDP dépend directement de l'arité de la variable d'état X_p ; c'est d'ailleurs la raison d'être de notre algorithme de réduction automatique d'arité de variables d'état.

Algorithme 8 : Fonction `PartitionneSousMDP`

Données : $\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle_{e \in \mathcal{K}_p}$ (liste de MDP énumérés)
Résultat : $\tilde{V}_p$ (partitionnement de V_p)
début
  $\langle V_p, \mathcal{A}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \rangle$: ***MDP énuméré sous-jacent de*** X_p;
  $\overline{\mathcal{T}} \leftarrow \emptyset$, $\overline{\mathcal{R}} \leftarrow \emptyset$;
  pour $e \in \mathcal{K}_p$ **faire**
    $\overline{\mathcal{T}} \leftarrow \overline{\mathcal{T}} \cup \mathcal{T}_e$;
    $\overline{\mathcal{R}} \leftarrow \overline{\mathcal{R}} \cup \mathcal{R}_e$;
  $|\tilde{V}_p| \leftarrow$ `CalculeDimensionPartitionnement()`;
  $\tilde{V}_p \leftarrow$ `Partitionne` $\left(\langle V_p, \mathcal{A}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \rangle, |\tilde{V}_p| \right)$;
  retourner $\tilde{V}_p$;
fin

L'algorithme de [25] appelle l'algorithme de [73] sur le graphe des transitions du MDP, après avoir mis le MDP sous forme de graphe. Un partitionnement en $|\tilde{V}_p|$ régions équilibrées (c'est-à-dire contenant toutes le même nombre d'états) est calculé approximativement. La complexité de cet algorithme (cf. [73]) est $O\left(3^{|\tilde{V}_p|}|V_p| \log |V_p|\right)$. C'est donc, dans notre cas, la complexité de la fonction `Partitionne` puisque tous nos MDP énumérés sont directement représentés sous forme de graphes. De toute façon, le coût de l'écriture du MDP sous forme de graphe est négligeable devant le coût du partitionnement automatique. Ainsi, la complexité de la fonction `PartitionneSousMDP` est :

$$O\left(2(C^d + 1) + 3^{|\tilde{V}_p|}|V_p| \log |V_p|\right) \tag{4.1}$$

En effet, une instanciation de $\mathcal{K}_p$ comprend d variables, et C est le cardinal du plus grand ensemble de variables d'état.

4.4 Fonction `DecomposeSousMDP`

Cette fonction décompose un sous-MDP énuméré $\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle$ en un sous-MDP abstrait $\langle \tilde{V}_p, \tilde{\mathcal{A}}_e, \tilde{\mathcal{T}}_e, \tilde{\mathcal{R}}_e \rangle$, connaissant le partitionnement $\tilde{V}_p$. Cette décomposition peut être réalisée de manière efficace grâce à l'algorithme de [24] (cf. section 2.1 page 29 et algorithme 9). Rappelons brièvement le fonctionnement de cet algorithme :

1. fonction `CalculeBornesVF` : calcule les bornes de la fonction de valeur sur l'espace d'état du sous-MDP énuméré $\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle$
   Complexité : $O\left(2|V_p||\mathcal{A}|\right)$
2. Pour chaque région $\tilde{v}_p$ du partitionnement $\tilde{V}_p$:
   (a) Initialise le cache de politiques locales $\Pi_{\tilde{v}_p}$ en calculant la politique locale optimale pour des valeurs périphériques $(V_{min}, \cdots, V_{min})$ avec la fonction `CalculePolitiqueLocale`
   Complexité : $O\left(\frac{\ln \epsilon(1-\gamma)}{\ln \gamma}|\mathcal{A}||\tilde{v}_p|^2\right)$
   (b) Initialise le cache de fonctions de valeur locales $f_{\tilde{v}_p}$ en évaluant la politique locale précédemment calculée pour des valeurs périphériques $\{(1, 0, \cdots, 0), \cdots, (0, \cdots, 0, 1)\}$ avec la fonction `EvaluePolitique`
   Complexité : $O\left(\frac{\ln \epsilon(1-\gamma)}{\ln \gamma}|Sper(\tilde{v}_p)||\mathcal{A}||\tilde{v}_p|^2\right)$
   (c) Répète tant qu'il existe un jeu de valeurs périphériques pour laquelle aucune des politiques locales du cache n'est optimale (donc autant de fois que le nombre $N_{\tilde{v}_p}$ de politiques locales générées dans la région $\tilde{v}_p$) :
      i. Pour chaque état de la région $\tilde{v}_p$, pour chaque action et chaque politique locale du cache (courant) : trouve par programmation linéaire un jeu de valeurs périphériques pour laquelle aucune des politiques locales du cache n'est optimale
      Complexité : $O\left(\mathcal{P}\left(|Sper(\tilde{v}_p)|, |Sper(\tilde{v}_p)| + |Eper(\tilde{v}_p)| + |\Pi_{\tilde{v}_p}|\right)\right)$
      en moyenne, où $\mathcal{P}$ est un polynôme (cf. [58])
      ii. Calcule la politique locale optimale pour les valeurs périphériques qui maximisent les solutions des programmes linéaires précédents avec la fonction `CalculePolitiqueLocale` (erreur de Bellman maximum sur chaque état de la région $\tilde{v}_p$, chaque action et chaque politique locale du cache)
      Complexité : $O\left(\frac{\ln \epsilon(1-\gamma)}{\ln \gamma}|\mathcal{A}||\tilde{v}_p|^2\right)$
      iii. Évalue la politique locale précédemment calculée pour des valeurs périphériques $\{(1, 0, \cdots, 0), \cdots, (0, \cdots, 0, 1)\}$ avec la fonction `EvaluePolitique`
      Complexité : $O\left(\frac{\ln \epsilon(1-\gamma)}{\ln \gamma}|Sper(\tilde{v}_p)||\mathcal{A}||\tilde{v}_p|^2\right)$

Algorithme 9 : Fonction `DecomposeSousMDP` [24]

Données : $\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle$ (sous-MDP énuméré par rapport à X_p), $\tilde{V}_p$ (partitionnement de V_p)
Résultat : $\langle \tilde{V}_p, \tilde{\mathcal{A}}_e, \tilde{\mathcal{T}}_e, \tilde{\mathcal{R}}_e \rangle$ (MDP énuméré décomposé)
début
  $\tilde{\mathcal{A}}_e \leftarrow \emptyset$, $\tilde{\mathcal{T}}_e \leftarrow \emptyset$, $\tilde{\mathcal{R}}_e \leftarrow \emptyset$;
  $(VF_{min}, VF_{max}) \leftarrow$ `CalculeBornesVF`$(\langle V_p, \mathcal{A}, \mathcal{T}_e, \mathcal{R}_e \rangle)$;
  pour $\tilde{v}_p \in \tilde{V}_p$ **faire**
    $\Pi_{\tilde{v}_p} \leftarrow \emptyset$, $F_{\tilde{v}_p} \leftarrow \emptyset$;
    $\pi^+ \leftarrow$ `CalculePolitiqueOptimale`$(VF_{min}, \cdots, VF_{min})$;
    $f_{\pi^+} \leftarrow$ `EvaluePolitique`(π^+);
    $\Pi_{\tilde{v}_p} \leftarrow \Pi_{\tilde{v}_p} \cup \{\pi^+\}$, $F_{\tilde{v}_p} \leftarrow F_{\tilde{v}_p} \cup \{f_{\pi^+}\}$;
    $Sortir \leftarrow$ **false**;
    répéter
      pour $v_p \in \tilde{v}_p$ **faire**
        pour $a \in \mathcal{A}$ **faire**
          pour $\pi \in \Pi_{\tilde{v}_p}$ **faire**
            Résoudre problème linéaire :

Maximiser :

$$\mathcal{R}_e(v_p, a) + \sum_{v'_p \in \tilde{v}_p} T_e^a(v_p, v'_p) f_\pi(v'_p, \mathcal{S}) - f_\pi(v_p, \mathcal{S})$$

Sujet à :

$$f_{\pi'}(e, \mathcal{S}) \leqslant f_\pi(e, \mathcal{S}) \ , \ \forall\, e \in Eper(\tilde{v}_p) \ , \ \forall\, \pi' \in \Pi_{\tilde{v}_p}$$
$$\mathcal{S}[i] \leqslant VF_{max} \ , \ 1 \leqslant i \leqslant |Sper(\tilde{v}_p)|$$

            $\longrightarrow \left(\mathcal{E}_{v_p,a,\pi}, \mathcal{S}_{v_p,a,\pi}\right)$
      si $\max\limits_{v_p \in \tilde{v}_p, a \in \mathcal{A}, \pi \in \Pi_{\tilde{v}_p}} \mathcal{E}_{v_p,a,\pi} > \epsilon$ **alors**
        $\pi^+ \leftarrow$ `CalculePolitiqueOptimale` $\left(\operatorname{argmax}\limits_{v_p \in \tilde{v}_p, a \in \mathcal{A}, \pi \in \Pi_{\tilde{v}_p}} \mathcal{S}_{v_p,a,\pi} \right)$;
        $f_{\pi^+} \leftarrow$ `EvaluePolitique`(π^+);
        $\Pi_{\tilde{v}_p} \leftarrow \Pi_{\tilde{v}_p} \cup \{\pi^+\}$; , $F_{\tilde{v}_p} \leftarrow F_{\tilde{v}_p} \cup \{f_{\pi^+}\}$;
      sinon
        $Sortir \leftarrow$ **true**;
    jusqu'à $Sortir =$ **true**;
fin

En notant $M_{\tilde{v}_p}$ le nombre politiques locales générées dans chaque région $\tilde{v}_p$ du partitionnement $\tilde{V}_p$, la complexité totale de la fonction `DecomposeSousMDP` est :

$$2|V_p||\mathcal{A}| + \sum_{\tilde{v}_p \in \tilde{V}_p} \Bigg(\frac{\ln \epsilon(1-\gamma)}{\ln \gamma} |\mathcal{A}||\tilde{v}_p|^2 \left(1 + |Sper(\tilde{v}_p)|\right) + \sum_{|\Pi_{\tilde{v}_p}|=1}^{M_{\tilde{v}_p}} \left(|\tilde{v}_p||\mathcal{A}||\Pi_{\tilde{v}_p}|\mathcal{P}\left(|Sper(\tilde{v}_p)|, |Sper(\tilde{v}_p)| + |Eper(\tilde{v}_p)| + |\Pi_{\tilde{v}_p}|\right) + |\tilde{v}_p||\mathcal{A}||\Pi_{\tilde{v}_p}| + \frac{\ln \epsilon(1-\gamma)}{\ln \gamma} |\mathcal{A}||\tilde{v}_p|^2 \left(1 + |Sper(\tilde{v}_p)|\right)\right)\Bigg)$$

Or, l'algorithme de partitionnement automatique [73] calcule un partitionnement équilibré, c'est-à-dire :

$$\forall\ \tilde{v}_p^1 \neq \tilde{v}_p^2\ ,\ |\tilde{v}_p^1| = |\tilde{v}_p^2| = \frac{|V_p|}{|\tilde{V}_p|}$$

De plus : $\forall\ \tilde{v}_p \in \tilde{V}_p\ ,\ |Eper(\tilde{v}_p)| \leqslant |\tilde{v}_p| = \frac{|V_p|}{|\tilde{V}_p|}$ et $|Sper(\tilde{v}_p)| \lesssim |\tilde{V}_p|$
La complexité est donc :

$$2|V_p||\mathcal{A}| + \frac{\ln \epsilon(1-\gamma)}{\ln \gamma} |\mathcal{A}| \frac{|V_p|^2}{|\tilde{V}_p|} \left(1 + |\tilde{V}_p|\right) + \frac{|V_p|}{|\tilde{V}_p|} |\mathcal{A}| \sum_{\tilde{v}_p \in \tilde{V}_p} \sum_{k=1}^{M_{\tilde{v}_p}} k\mathcal{P}\left(|\tilde{V}_p|, |\tilde{V}_p| + \frac{|V_p|}{|\tilde{V}_p|} + k\right) + \frac{|V_p|}{|\tilde{V}_p|} |\mathcal{A}| \sum_{\tilde{v}_p \in \tilde{V}_p} \frac{M_{\tilde{v}_p}\left(M_{\tilde{v}_p}+1\right)}{2} + \frac{\ln \epsilon(1-\gamma)}{\ln \gamma} |\mathcal{A}| \frac{|V_p|^2}{|\tilde{V}_p|} \left(1 + |\tilde{V}_p|\right) \sum_{\tilde{v}_p \in \tilde{V}_p} M_{\tilde{v}_p}$$

Supposons que les politiques locales générées au total, c'est-à-dire l'ensemble $\tilde{\mathcal{A}}$ des macro-actions du MDP factorisé abstrait, sont bien réparties dans les régions :

$$\forall\ \tilde{v}_p \in \tilde{V}_p\ ,\ M_{\tilde{v}_p} = \frac{\tilde{\mathcal{A}}}{|\tilde{V}_p|}$$

La complexité est alors :

$$2|V_p||\mathcal{A}| + \frac{\ln \epsilon(1-\gamma)}{\ln \gamma} |\mathcal{A}| \frac{|V_p|^2}{|\tilde{V}_p|} \left(1 + |\tilde{V}_p|\right) + |V_p||\mathcal{A}| \sum_{k=1}^{\frac{|\tilde{\mathcal{A}}|}{|\tilde{V}_p|}} k\mathcal{P}\left(|\tilde{V}_p|, |\tilde{V}_p| + \frac{|V_p|}{|\tilde{V}_p|} + k\right) + \frac{|V_p|}{|\tilde{V}_p|} |\mathcal{A}| \frac{|\tilde{\mathcal{A}}|}{2} \left(\frac{|\tilde{\mathcal{A}}|}{|\tilde{V}_p|} + 1\right) + \frac{\ln \epsilon(1-\gamma)}{\ln \gamma} |\mathcal{A}||\tilde{\mathcal{A}}| \frac{|V_p|^2}{|\tilde{V}_p|^2} \left(1 + |\tilde{V}_p|\right)$$

Lorsque les données sont grandes, la complexité est donc :

$$O\left(|\mathcal{A}||\tilde{\mathcal{A}}|\frac{|V_p|}{|\tilde{V}_p|}\left(\frac{\ln\epsilon(1-\gamma)}{\ln\gamma}|V_p|+\frac{|\tilde{\mathcal{A}}|}{|\tilde{V}_p|}\right)+|V_p||\mathcal{A}|\sum_{k=1}^{\frac{|\tilde{\mathcal{A}}|}{|\tilde{V}_p|}}k\mathcal{P}\left(|\tilde{V}_p|,|\tilde{V}_p|+\frac{|V_p|}{|\tilde{V}_p|}+k\right)\right)$$

où $\mathcal{P}$ est un polynôme donné dans [58].

4.5 Fonction GenereMDPabstrait

Cette fonction génère le MDP abstrait factorisé final en s'appuyant sur le théorème 9 page 82. L'algorithme 10 correspondant renvoie aux fonctions :

- `CalculeArbreProbabiliteVariableReduite` : génère l'arbre de probabilité de la variable $\tilde{X}_p$ pour une politique π donnée et l'instanciation correspondante $e = \Pi(\pi)$
- `CalculeProbabilitesAgregees` : calcule les probabilités agrégées de transition (sous formes d'ADD) pour les variables d'état X_i, $i \neq p$, en s'appuyant sur le théorème 10 page 87 pour une politique π donnée et l'instanciation correspondante $e = \Pi(\pi)$
- `CalculeArbreProbabiliteTransitionsLocales` : génère, pour une politique π donnée et pour l'instanciation correspondante $e = \Pi(\pi)$, l'arbre de probabilité (sous forme d'ADD) des transitions locales à la région de V_p où π est définie
- `CalculeArbreProbabiliteAutresVariables` : génère l'arbre de probabilité (sous forme d'ADD) d'une variable d'état X_i, $i \neq p$, pour une politique π donnée et l'instanciation correspondante $e = \Pi(\pi)$
- `CalculeArbreRecompense` : génère l'arbre de récompense de la macro-action π pour l'instanciation correspondante $e = \Pi(\pi)$

Afin d'illustrer le fonctionnement de cet algorithme, nous reprenons dans la suite l'exemple du MDP factorisé défini dans la figure 3.6 dont la variable X est réduite. Nous rappelons que les sous-MDP énumérés par rapport à X par la fonction `EnumereSousMDP` (cf. algorithme 7) sont représentés dans la figure 3.7. Nous supposons enfin que le partitionnement de l'espace d'états des sous-MDP énumérés par la fonction `PartitionneSousMDP` (cf. algorithme 8) produit 2 macro-états $\tilde{x}_1$ et $\tilde{x}_2$.

Algorithme 10 : Fonction `GenereMDPabstrait`

Données : $\langle \tilde{V}_p, \tilde{\mathcal{A}}_e, \tilde{\mathcal{T}}_e, \tilde{\mathcal{R}}_e, \rangle_{e \in \mathcal{K}_p}$ (liste de MDP énumérés abstraits), $\mathcal{T}$ (transitions factorisées), p
Résultat : $\langle \tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$ (MDP factorisé abstrait)
début
  $\tilde{\mathcal{V}} \leftarrow \emptyset$, $\tilde{\mathcal{A}} \leftarrow \emptyset$, $\tilde{\mathcal{T}} \leftarrow \emptyset$, $\tilde{\mathcal{R}} \leftarrow \emptyset$;
  pour $i \neq p$ **faire**
    $\tilde{\mathcal{V}} \leftarrow \tilde{\mathcal{V}} \cup \{V_i\}$;
  $\tilde{\mathcal{V}} \leftarrow \tilde{\mathcal{V}} \cup \left\{\tilde{V}_p\right\}$;
  pour $e \in \mathcal{K}_p$ **faire**
    $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \left\{\tilde{\mathcal{A}}_e\right\}$;
    pour $\pi \in \tilde{\mathcal{A}}_e$ **faire**
      $\tilde{T}_p^{\pi}$: ***Arbre de probabilité***;
      $\tilde{T}_p^{\pi} \leftarrow$ `CalculeArbreProbabiliteVariableReduite` $\left(e, \tilde{\mathcal{T}}_e^{\pi}, p\right)$;
      $\tilde{\mathcal{T}}[\pi][p] \leftarrow \tilde{T}_p^{\pi}$;
      T^{π} : ***transitions factorisées agrégées***;
      bdd_e : ***BDD***;
      $(T^{\pi}, bdd\_e) \leftarrow$ `CalculeProbabilitesAgregees`$(e, \pi, \mathcal{T})$;
      $add\mathcal{T}_e^{\pi}$: ***ADD***;
      $add\mathcal{T}_e^{\pi} \leftarrow$
      `CalculeArbreProbabiliteTransitionsLocales`$(bdd\_e, \mathcal{T}_e^{\pi})$;
      pour $i \neq p$ **faire**
        $\tilde{T}_i^{\pi}$: ***ADD***;
        $\tilde{T}_i^{\pi} \leftarrow$
        `CalculeArbreProbabiliteAutresVariables` $(e, T^{\pi}, add\mathcal{T}_e^{\pi}, i, p)$;
        $\tilde{\mathcal{T}}[\pi][i] \leftarrow \tilde{T}_i^{\pi}$;
      $\tilde{R}^{\pi}$: ***Arbre de récompense***;
      $\tilde{R}^{\pi} \leftarrow$ `CalculeArbreRecompense` $\left(e, \tilde{\mathcal{R}}_e, p\right)$;
      $\tilde{R}[\pi] \leftarrow \tilde{R}^{\pi}$;
  retourner $\langle \tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$;
fin

4.5.1 Fonction CalculeArbreProbabiliteVariableReduite

D'après le théorème 9 page 82 :

$$\forall \left(\pi, \bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p\right) \in \tilde{\mathcal{A}} \times \left(\bigotimes_{j\neq p} V_j\right)^2 \times \tilde{V}_p \ , \ \forall\, \tilde{v}'_p \in \tilde{V}_p \ :$$

$$\tilde{T}_p^{\pi}\left(\bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p\right)(\tilde{v}'_p) = \begin{cases} 0 & \text{si } \bigwedge\limits_{j\neq p} v_j \bigwedge\limits_{j\neq p} v'_j \notin \Pi(\pi) \\ \tilde{\mathcal{T}}^{\pi}_{\bigwedge\limits_{j\neq p} v_j \bigwedge\limits_{j\neq p} v'_j}\left(\tilde{v}_p, \tilde{v}'_p\right) & \text{sinon} \end{cases}$$

Ainsi, si l'arbre de probabilité de la variable $\tilde{X}_p$ est construit en commençant par les variables de l'instanciation $e = \Pi(\pi)$ (qui est une liste de variables valuées), seul le sous-arbre correspondant au chemin e depuis la racine de l'arbre est non nul. Ce sous-arbre dépend a priori de la variable $\tilde{X}_p$ qui est donc rajoutée à la fin du chemin e. La feuille correspondant à chaque valeur de $\tilde{v}_p$ de $\tilde{X}_p$ est une liste de probabilités d'obtenir les différentes valeurs $\tilde{v}'_p$ de $\tilde{X}'_p$ connaissant e et $\tilde{v}_p$, c'est-à-dire $\tilde{\mathcal{T}}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(\tilde{v}_p, \tilde{v}'_p\right)$ d'après le résultat ci-dessus. Ce procédé est décrit dans l'algorithme 11.

La complexité de cette fonction est :

$$O\left(2 + dC\left(1 + \max\left(1 + \max\left(2 + |\tilde{V}_p|\left(2 + |\tilde{V}_p|\right)\right), 2\right), 1 + |\tilde{V}_p|\right)\right)$$

où :

- d est la longueur d'une instanciation de $\mathcal{K}_p$ (c'est-à-dire la taille de la liste e)
- C est le cardinal du plus grand ensemble de variables d'état

Si $|\tilde{V}_p|$ est grand, la complexité devient : $0\left(dC|\tilde{V}_p|^2\right)$

A titre d'exemple, considérons une politique locale π obtenue en décomposant le sous-MDP énuméré $\langle V_X, \mathcal{A}, \tilde{\mathcal{T}}_{(z_1,y'_1)}, \tilde{\mathcal{R}}_{(z_1,y'_1)}\rangle$, **définie dans la région** $\tilde{x}_1$. L'instanciation correspondant à cette politique est donc $e = \Pi(\pi) = (z_1, y'_1)$. L'arbre de probabilité de la variable $\tilde{X}$ est donné dans la figure 4.1. Les feuilles où la politique π n'est pas définie (états complémentaires de $(z_1, y'_1, \tilde{x}_1)$) sont des listes nulles de taille $|\tilde{X}|$: *ce ne sont donc pas des lois de probabilité.* Cette représentation n'a pas de sens mathématique, mais elle ne modifie pas la solution du problème si un arbre de masque est utilisé, tel que toutes les feuilles sont nulles sauf celle correspondant à l'état $(z_1, y'_1, \tilde{x}_1)$ qui vaut 1 (cf. section 3.5 et section 2.2 page43).

Ainsi, l'arbre obtenu est donc extrêmement creux puisqu'**une seule feuille est non nulle** (ce résultat est toujours vrai dans le cas général). Il est nettement plus compact que l'arbre originel de la variable X représenté dans la figure 3.6.

Algorithme 11 : Fonction `CalculeArbreProbabiliteVariableReduite`

Données : e (liste d'instanciation des variables de $\mathcal{K}_p$), $\tilde{\mathcal{T}}_e^{\pi}$ (liste de transitions énumérées), p
Résultat : $\tilde{T}_p^{\pi}$ (arbre de probabilité de $\tilde{X}_p$)
début
  $\tilde{T}_p^{\pi}.racine() \leftarrow e.premier$;
  $noeud \leftarrow \tilde{T}_p^{\pi}.racine()$;
  pour $v \in e$ **faire**
    pour $vv \in v.variable()$ **faire**
      si $vv = v.valeur()$ **alors**
        si $v = e.dernier()$ **alors**
          $noeud.fils(vv) \leftarrow \tilde{X}_p$;
          $noeud \leftarrow noeud.fils(vv)$;
          pour $\tilde{v}_p \in \tilde{V}_p$ **faire**
            lp : ***liste de probabilités***;
            $lp \leftarrow \emptyset$;
            pour $\tilde{v}'_p \in \tilde{V}_p$ **faire**
              $lp \leftarrow lp \cup \left\{\tilde{\mathcal{T}}_e^{\pi}\left(\tilde{v}_p, \tilde{v}'_p\right)\right\}$;
            $noeud.fils(\tilde{v}_p) \leftarrow feuille(lp)$;
        sinon
          $noeud.fils(vv) \leftarrow e.suivant(v)$;
          $noeud \leftarrow noeud.fils(vv)$;
      sinon
        lr : ***liste de réels***;
        pour $1 \leqslant k \leqslant |\tilde{V}_p|$ **faire**
          $lr \leftarrow lr \cup \{0\}$;
        $noeud.fils(vv) \leftarrow feuille(lr)$;
  retourner $\tilde{T}_p^{\pi}$;
fin

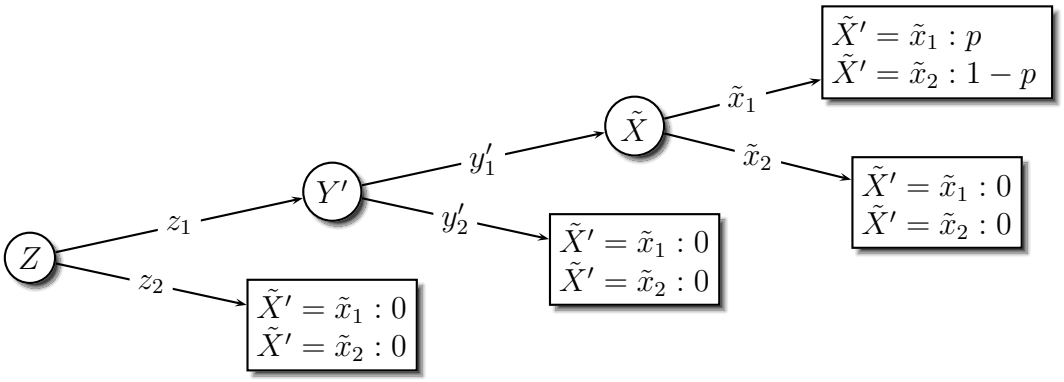


FIG. 4.1 – Arbre de probabilité de $\tilde{X}$ pour une politique locale π du sous-MDP énuméré $\langle V_X, \mathcal{A}, \tilde{\mathcal{T}}_{(z_1,y'_1)}, \tilde{\mathcal{R}}_{(z_1,y'_1)} \rangle$ (cf. figure 3.7)

Fonction `CalculeProbabilitesAgregees`

D'après le théorème 10 :

$$\left[\begin{array}{c} T_1^\pi \left(\bigwedge_{1 \leqslant j \leqslant n} v_j, \bigwedge_{j \neq 1} v'_j \right)(v'_1) \\ \vdots \\ T_n^\pi \left(\bigwedge_{1 \leqslant j \leqslant n} v_j, \bigwedge_{j \neq n} v'_j \right)(v'_n) \end{array} \right]_{(v,v') \in e \times \mathcal{V}} = \lim_{t \to +\infty} u_t$$

où u_t est la liste de $\mathbb{R}^{n|e||\mathcal{V}|}$ définie récursivement par :

– $\forall\, t \in \mathbb{N}^*\ ,\ \forall\, (v,v') \in e \times \mathcal{V}\ ,\ \forall\, 1 \leqslant i \leqslant n\ ,$

$$u_{t+1}(v,v',i) = \sum_{v'' \in e \cup V_p} T_i^{\pi(v''_p)} \left(\bigwedge_{1 \leqslant j \leqslant n} v''_j, \bigwedge_{j \neq i} v'_j \right)(v'_i) \prod_{k=1}^{n} u_t(v,v'',k)$$

– $\forall\, (v,v') \in e \times \mathcal{V}\ ,\ \forall\, 1 \leqslant i \leqslant n\ , u_0(v,v',i) = \left\{ \begin{array}{l} 1 \text{ si } v = v' \\ 0 \text{ sinon} \end{array} \right.$

Pour un t donné, u_t est une matrice $\mathcal{M}^{n \times e \times \mathcal{V}}$ de dimension potentiellement très grande. Il n'est donc pas question de représenter u_t sous forme de liste ou de tableau multidimensionnel. Comme pour la fonction de valeur des MDP factorisés [26], nous proposons de représenter u_t sous une forme factorisée, sous forme d'arbre de décision [26] ou d'ADD [69]. Pour chaque t, $u_t(v,v',i)$ représente la probabilité que $X_i^t = v'_i$ sachant que la politique π est suivie, que $(X_1^0 = v_1, \cdots, X_n^0 = v_n)$ et que $\left(X_1^t = v'_1, \cdots, X_{i-1}^t = v'_{i-1}, X_{i+1}^t = v'_{i+1}, \cdots, X_n^t = v'_n\right)$. Nous pouvons donc représenter u_t sous forme d'une liste d'arbre (ou d'ADD) de probabilité pour chaque variable X_i. La fonction `CalculeProbabilitesAgregees` met à jour itérativement

les arbres de probabilité de la suite $(u_t)_{t\in\mathbb{N}}$ de la même manière que la version factorisée de l'opérateur de Bellman (cf. [26, 31]).

Dans l'algorithme 12, nous représentons les arbres de probabilité de la suite $(u_t)_{t\in\mathbb{N}}$ sous forme d'ADD car :

- les calculs algébriques sur des structures factorisées sont plus efficaces avec des ADD qu'avec des arbres de décision [69, 26] ;
- le MDP abstrait factorisé sera ensuite résolu en utilisant des ADD (pour la même raison).

Ceci nécessite donc de transformer initialement tous les arbres de probabilité du problème en ADD. Nous ne prenons pas en compte la complexité de cette transformation car, de toute façon, elle est nécessaire pour résoudre le MDP factorisé initial efficacement (avant réduction de l'arité de X_p). La transformation inverse n'est utile que dans le cas où l'opérateur humain souhaite connaître le modèle du MDP factorisé abstrait. Elle n'est pas nécessaire à l'optimisation du MDP abstrait factorisé si bien qu'elle peut être considérée comme une option de l'algorithme. Ainsi, la transformation inverse ne sera pas non plus intégrée dans le calcul de la complexité globale de notre algorithme de réduction automatique de l'arité d'une variable d'état. Précisons enfin que les fonction de transformation directe et inverse d'arbre de décision en ADD sont détaillées dans la section 2.2 (page 43).

Pour les mêmes raisons, nous supposons que les variables du problème sont binaires. Nous appelons $\left(X_p^k\right)_{1\leqslant k\leqslant(E_{\mathbb{N}}(\log_2(|V_p|))+1)}$ les $(E_{\mathbb{N}}(\log_2(|V_p|))+1)$ variables *binaires* résultant de la transformation de X_p en variable binaires.

L'algorithme 12 repose sur les fonctions de manipulation des BDD [68] et des ADD [69]. Les fonctions utilisées sont toutes implémentées dans la librairie informatique CUDD [70]. Afin de comprendre l'algorithme 12, il est nécessaire de rappeler que les BDD encodent des fonctions booléennes à valeur booléenne, et que les ADD représentent des fonctions booléennes à valeur dans $\mathbb{R}$ (cf. section 2.2 page 43). Les étapes de l'algorithme sont :

1. Calcul du BDD *bdd_e* représentant e, qui correspond à la fonction caractéristique $\chi_{\left(X_{i_1}=v_{i_1},\cdots,X_{i_d}=v_{i_d}\right)}$ où $(v_{i_1},\cdots,v_{i_d})$ sont les valeurs des variables de l'instanciation e. Nous utilisons pour cela la fonction $ITE(f,g,h)=f\cdot g+\overline{f}\cdot h$.
2. Calcul des ADD *add_t* représentant les transitions factorisées de la politique π pour chaque variable d'état. La politique est un vecteur de $|\mathcal{A}|$ BDD qui représentent chacun l'ensemble d'état où l'action a de la politique s'applique : $\pi[a]=\chi_{\pi^{-1}(a)}$.
3. Construction du cube *cube_sommation* qui permet de sommer, dans un ADD, tous les sous-arbres des variables présentes dans le cube ; ceci permet donc de calculer la somme $\sum_{v''\in e\cup V_p}(\cdots)$ où v'' est une liste de variables instanciées.
4. Itération sur les ADD de probabilités agrégées :

Algorithme 12 : Fonction CalculeProbabilitesAgregees

Données : e (liste d'instanciation des variables de $\mathcal{K}_p$), π (politique locale), $\mathcal{T}$ (transitions factorisées)
Résultat : T^{π} (transitions factorisées agrégées)
début
  $bdd\_e \leftarrow$ `ReadOne`();
  pour $v \in e$ **faire**
    si $v.valeur() =$ **true alors**
      $bdd\_e \leftarrow$ `bddAnd`(bdd_e, `ITE`($v.variable()$, `ReadOne`(), `ReadLogicZero`()));
    sinon
      $bdd\_e \leftarrow$ `bddAnd`(bdd_e, `ITE`($v.variable()$, `ReadLogicZero`(), `ReadOne`()));
  $add\_t, U, \overline{U}$: ***vecteurs de*** n ***ADD***;
  pour $1 \leqslant i \leqslant n$ **faire**
    $add\_t[i] \leftarrow$ `ReadZero`();
    pour $a \in \mathcal{A}$ **faire**
      $add\pi[a] \leftarrow$ `BddToAdd`($\pi[a]$);
      $add\_temp \leftarrow$ `addApply`(`addTimes`, T^a, $add\pi[a]$);
      $add\_t[i] \leftarrow$ `addApply`(`addPlus`, $add\_t[i]$, add_temp);
  $cube\_sommation \leftarrow$ `bddCube` $\left(e \wedge \left(\bigwedge_{1 \leqslant k \leqslant (E_{\mathbb{N}}(\log_2(|V_p|))+1)} \left\{X_p^k\right\}\right)\right)$;
  pour $1 \leqslant i \leqslant n$ **faire**
    $\overline{U}[i] \leftarrow$ `BddToAdd`(bdd_e);
  répéter
    $add\_temp \leftarrow$ `ReadOne`();
    pour $1 \leqslant i \leqslant n$ **faire**
      $U[i] \leftarrow \overline{U}[i]$;
      $add\_temp \leftarrow$ `addApply`(`addTimes`, $U[i]$, add_temp);
    pour $1 \leqslant i \leqslant n$ **faire**
      $\overline{U}[i] \leftarrow$ `addApply`(`addTimes`, $add\_t[i]$, add_temp);
      $\overline{U}[i] \leftarrow$ `addExistAbstract`($\overline{U}[i]$, $cube\_sommation$);
      $\overline{U}[i] \leftarrow$ `addPermute`($\overline{U}[i]$, $PrimedToUnprimed$);
  jusqu'à $\max_{1 \leqslant i \leqslant n} \left\{\text{`addSupNorm`}(\text{`addApply`}(\text{`addMinus`}, \overline{U}[i], U[i]))\right\} < \epsilon$;
  pour $1 \leqslant i \leqslant n$ **faire**
    $\overline{U}[i] \leftarrow$ `addPermute`($\overline{U}[i]$, $UnprimedToPrimed$);
    $\overline{U}[i] \leftarrow$ `addApply`(`addTimes`, $\overline{U}[i]$, `BddToAdd`(bdd_e));
  retourner $\overline{U}$;
fin

(a) Calcul de l'ADD produit *add_temp* des précédentes probabilités agrégées, soit $\prod_{k=1}^{n} u_t(v, v'', k)$; les probabilités agrégées sont des fonctions de v'' (en tant qu'ADD) donc l'ADD produit obtenu est également une fonction de v'' (toutes les valeurs possibles de v'' sont prises en compte dans l'ADD produit)

(b) Multiplication de l'ADD produit *add_temp* des précédentes probabilités agrégées par l'ADD des probabilités de transition pour la politique π (*add_t*)

(c) Sommation sur toutes les valeurs des variables $v'' \cup V_p$ à l'aide du cube *cube_sommation*

(d) La probabilité obtenue est exprimée en fonction des variables primées (ou post-action), il faut donc transformer les variables primées en variables non primées (pré-action)

5. Les probabilités agrégées obtenues $\overline{U}$ représentent les probabilités d'obtention des différentes valeurs des variables d'état après l'application de la politique π. Nous devons donc primer les variables non primées des ADD représentant les probabilités agrégées.

6. Les probabilités agrégées $\overline{U}$ sont définies uniquement sur l'ensemble d'états de l'instanciation e, si bien qu'il est nécessaire de les multiplier par le BDD *bdd_e* représentant l'instanciation e.

La complexité de la fonction `CalculeProbabilitesAgregees` n'est pas évidente à évaluer car les opérations sur les BDD et les ADD sont proportionnelles à la taille des graphes représentant les structures de données [68]. Le temps d'accès aux données est $O(\log N)$ où N est le nombre d'éléments non nuls stockées dans les données [69]. Ainsi, comme pour les matrices creuses [50, 21, 48, 49], les opérations sur les ADD sont d'autant plus efficaces que les données sont creuses, mais la densité en éléments des données n'est pas théoriquement évaluable. Néanmoins, les tests de [31] tendent à montrer que les ADD sont en général très efficaces dans le cadre des MDP. En effet, non seulement les matrices de transition sont très creuses, mais les fonctions algébriques (ADD) sont en général constantes sur de grands sous-espaces, ce qui résulte en des graphes de données relativement petits (chaque sous-espace correspond à un sous-ensemble de variables d'état instanciées). De plus, les ADD et les BDD permettent de regrouper systématiquement les sous-graphes identiques, ce qui réduit considérablement leur taille [31] par rapport aux arbres de décision [26]. Une étude détaillée de la complexité des BDD se trouve dans [74] et [75].

Dans le pire des cas, **extrêmement rare avec les BDD et les ADD**, les graphes de données sont pleins, c'est-à-dire que les variables sont toutes instanciées à tous les niveaux des graphes. Ce cas revient à utiliser un espace d'états énumérés et des matrices de transition pleines. Calculée à partir du théorème 10, la complexité

de la fonction CalculeProbabilitesAgregees est, dans ce cas pessimiste :

$$O\left(3nC^d|\mathcal{V}| + C^d\left|\frac{\ln\epsilon - \ln n\alpha}{\ln\beta}\right|\left(n|\mathcal{V}|\left(C^d+1\right)(n+1)\right)\right)$$

où :

- d est la longueur d'une instanciation de e
- C est le cardinal du plus grand ensemble de variables d'état
- $\alpha \geqslant 0$ et $0 \leqslant \beta < 1$ sont donnés par le théorème ergodique appliqué aux probabilités stationnaires dans les chaînes de Markov ergodiques [76, 77] :
  Soit $(v_t)_{t\in\mathbb{N}}$ la suite des produits des probabilités des variables d'état (suite $(u_t)_{t\in\mathbb{N}}$). Si la chaîne de Markov $(\mathcal{V}, \mathcal{T}^\pi)$ est ergodique, il existe $v^* \in \mathbb{N}^{|e||\mathcal{V}|}$, $\alpha \geqslant 0$ et $0 \leqslant \beta < 1$ tels que :

  $$\forall\, t \in \mathbb{N}\ ,\ \|v_t - v^*\| \leqslant \alpha\beta^t$$

  Ce théorème est l'équivalent du théorème de convergence de l'opérateur de Bellman stochastique pour la fonction de valeur des MDP [14]. Comme, à chaque instant t du processus, l'élément u_t est une somme finie de n éléments de v_t (cf. démonstration du théorème 10), la suite $(u_t)_{t\in\mathbb{N}}$ est ϵ-convergente si la suite $(v_t)_{t\in\mathbb{N}}$ est $\frac{\epsilon}{n}$-convergente.

Ainsi, si les données sont grandes, la complexité de l'algorithme 12 dans le pire cas, bien plus grande que la complexité moyenne, est :

$$O\left(n^2|\mathcal{V}|C^{2d}\left|\frac{\ln\epsilon - \ln n\alpha}{\ln\beta}\right|\right) \tag{4.2}$$

Donnons un exemple d'arbre de probabilité agrégée qui nous servira dans les paragraphes suivants. Considérons une politique locale π obtenue en décomposant le sous-MDP énuméré $\langle V_X, \mathcal{A}, \tilde{\mathcal{T}}_{(z_1,y'_1)}, \tilde{\mathcal{R}}_{(z_1,y'_1)}\rangle$, **définie dans la région** $\tilde{x}_1$. L'instanciation correspondant à cette politique est donc $e = \Pi(\pi) = (z_1, y'_1)$. Supposons que le partitionnement $\tilde{V}_X$ est : $\tilde{x}_1 = \{x_1, x_2, x_4\}$ et $\tilde{x}_2 = \{x_3\}$. La résolution analytique du système d'équation non linéaire donnant les probabilités agrégées est fastidieuse car il est de degré 3 (autant que de variables d'état). Ainsi, l'arbre de probabilité agrégée de la variable Z, présenté dans la figure 4.2, est purement fictif mais sa structure tient compte du réseau bayésien dynamique du MDP initial et du fait que la politique π n'est définie que sur $\tilde{x}_1$.

4.5.2 Fonction CalculeArbreProbabiliteTransitionsLocales

Les probabilités agrégées étant représentées sous forme d'ADD à la sortie de la fonction CalculeProbabilitesAgregees, les arbres de probabilité des variables

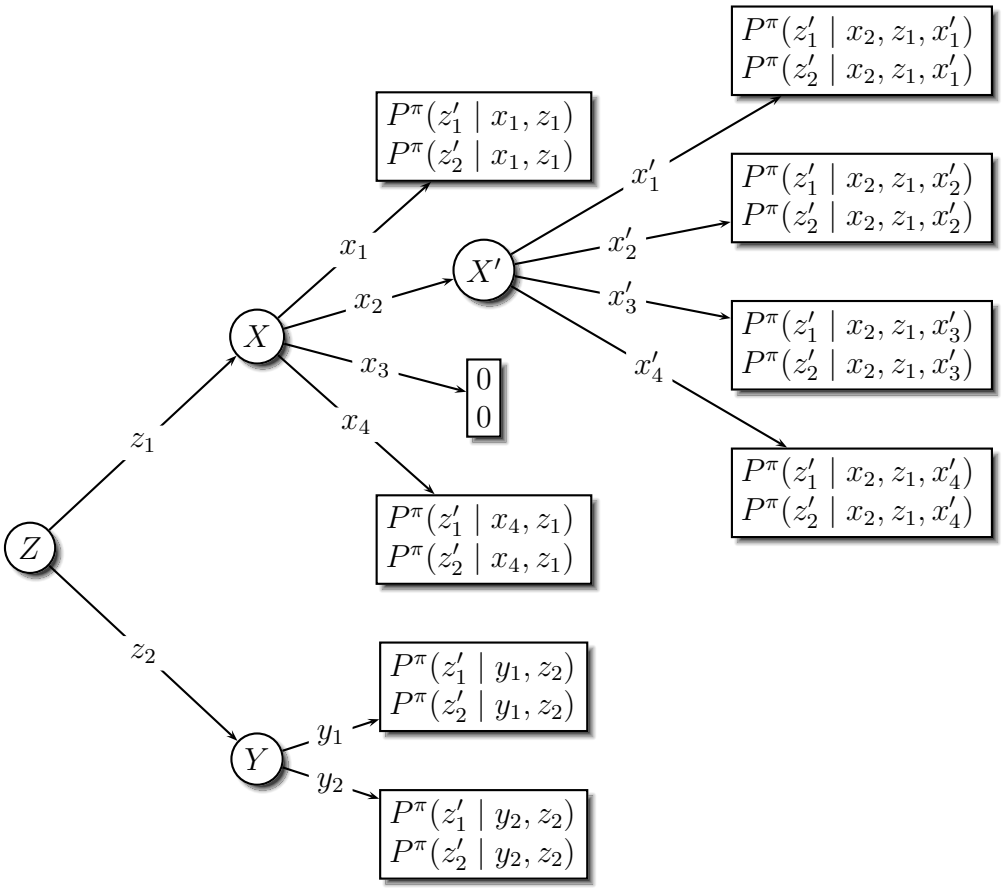


FIG. 4.2 – Arbre de probabilité agrégée de la variable Z pour une politique locale π du sous-MDP énuméré $\langle V_X, \mathcal{A}, \tilde{\mathcal{T}}_{(z_1,y'_1)}, \tilde{\mathcal{R}}_{(z_1,y'_1)} \rangle$ (cf. figure 3.7)

d'état autres que $\tilde{X}_p$ sont générés sous formes d'ADD par la fonction `CalculeArbreProbabiliteAutresVariables` (cf. algorithme 10). La formule donnant ces arbres de probabilité dans le théorème 9 montre que les arbres de probabilité de la variable $\tilde{X}_p$ doivent être convertis en ADD afin de pouvoir les multiplier par les ADD des probabilités agrégées. L'algorithme 13 transforme donc la fonction $\mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}\left(v_p, v'_p\right)$ en ADD.

L'utilisation des ADD requière que toutes les variables d'état soient binaires. Comme pour la fonction `CalculeProbabilitesAgregrees`, nous supposons que la variable X_p a été transformée en $E_{\mathbb{N}}\left(\log_2\left(|V_p|\right)\right) + 1$ variables binaires.
Nous appelons t la fonction qui à une valeur de la variable $|V_p|$-aire X_p associe le vecteur de valeurs des variables X_p^k correspondantes. Cette fonction est générée sous forme de vecteur lors de la transformation de X_p en variables binaires.

A titre d'exemple, la figure 4.3 représente l'arbre de probabilité $\mathcal{T}_e^{\pi}$ des transitions

Algorithme 13 : Fonction CalculeArbreProbabiliteTransitionsLocales

Données : bdd_e (BDD), T_e^π (transitions énumérées)
Résultat : $addT_e^\pi$ (ADD)
début
  $addT_e^\pi \leftarrow$ ReadZero();
  pour $v_p \in \tilde{v}_p$ **faire**
    $bdd\_vp \leftarrow$ ReadOne();
    pour $1 \leqslant k \leqslant E_{\mathbb{N}}(\log_2(|V_p|)) + 1$ **faire**
      si $t[v_p][k] =$ **true alors**
        $bdd\_vp \leftarrow$ bddAnd(bdd_vp, ITE(X_p^k, ReadOne(), ReadLogicZero()));
      sinon
        $bdd\_vp \leftarrow$ bddAnd(bdd_vp, ITE(X_p^k, ReadLogicZero(), ReadOne()));
    pour $v'_p \in \tilde{v}'_p$ **faire**
      $bdd\_vpp \leftarrow 1$;
      pour $1 \leqslant k \leqslant E(\log_2(|V_p|)) + 1$ **faire**
        si $t[v'_p][k] =$ **true alors**
          $bdd\_vpp \leftarrow$ bddAnd(bdd_vp, ITE(X'^k_p, ReadOne(), ReadLogicZero()));
        sinon
          $bdd\_vpp \leftarrow$ bddAnd(bdd_vp, ITE(X'^k_p, ReadLogicZero(), ReadOne()));
      $add\_vp\_vpp \leftarrow$ BddToAdd(bdd_vp, bdd_vpp);
      $add\_temp \leftarrow$ addApply(addTimes, $add\_vp\_vpp, T_e^\pi(v_p, v'_p)$);
      $addT_e^\pi \leftarrow$ addApply(addPlus, $addT_e^\pi, add\_temp$);
      $addT_e^\pi \leftarrow$ addApply(addTimes, $addT_e^\pi$, BddToAdd(bdd_e));
  retourner $addT_e^\pi$;
fin

locales d'une politique π définie pour l'instanciation $e = (z_1, y'_1)$ et dans la région $\tilde{x}_1 = \{x_1, x_2, x_4\}$.

4.5.3 Fonction `CalculeArbreProbabiliteAutresVariables`

D'après le théorème 9 :

$$\forall\, i \neq p\ ,\ \forall (\pi, v, v') \in \tilde{\mathcal{A}} \times \left(\bigotimes_{j\neq p} V_j\right) \times \left(\bigotimes_{j\neq\{i,p\}} V_j\right)\ ,\ \tilde{T}_i^{\pi}\left(\bigwedge_{j\neq p} v_j, \bigwedge_{j\neq\{i,p\}} v'_j, \tilde{v}_p, \tilde{v}'_p\right)(v'_i) =$$

$$\begin{cases} 0 \text{ si } \bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j \not\subset \Pi(\pi) \\ \dfrac{1}{|\tilde{v}_p|} \displaystyle\sum_{v_p \in \tilde{v}_p} \dfrac{\displaystyle\sum_{v'_p \in \tilde{v}'_p} \mathcal{T}_i^{\pi}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)(v'_i) \cdot \mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}(v_p, v'_p)}{\displaystyle\sum_{v'_p \in \tilde{v}'_p} \sum_{v'_{i,k} \in V_i} T_i^{\pi}\left(\bigwedge_{1\leqslant j\leqslant n} v_j, \bigwedge_{j\neq i} v'_j\right)\left(v'_{i,k}\right) \cdot \mathcal{T}^{\pi}_{\bigwedge_{j\neq p} v_j \bigwedge_{j\neq p} v'_j}(v_p, v'_p)} \text{ sinon} \end{cases}$$

La somme sur les états isolés d'une région est possible dans les arbres de décision (ou les ADD) à condition d'utiliser des masques sous forme d'arbre (ou d'ADD) qui représentent la région sur laquelle on somme. En effet, une région $\tilde{v}_p$ de la variable X_p correspond à un sous-ensemble de valeurs $\left(v_p^k\right)_{1\leqslant k\leqslant|\tilde{v}_p|}$ de la variable. Ainsi, il est possible de sommer un arbre donné sur ces valeurs uniquement, en multipliant cet arbre par un masque représentant ces valeurs puis en sommant dans l'arbre obtenu toutes les valeurs de la variable. Cette manipulation vaut également pour les ADD, mais ces derniers proposent la notion de «cubes» qui représentent un ensemble de variables binaires : il suffit alors de sommer directement sur le cube dont les variables représentent la région $\tilde{v}_p$ (cf. algorithme 14). Enfin, les régions du partitionnement ne se recouvrent pas si bien que l'arbre final, qui représente une fonction – entre autres – de toutes les valeurs de $\tilde{X}$ et $\tilde{X}'$, est obtenu en sommant les arbres correspondants à chaque région. Par conséquent, les sous-arbres correspondants à chaque région ne se recouvrent pas.

Contrairement aux arbres de probabilité de la variable abstraite $\tilde{X}_p$, dont une seule feuille est non nulle (cf. fonction `CalculeArbreProbabiliteVariableReduite`), les arbres de probabilité des autres variables d'état peuvent avoir plusieurs feuilles non nulles. En effet, ils sont obtenus à partir des arbres de probabilité agrégée (cf. fonction `CalculeProbabilitesAgregees`) qui sont définis sur le sous-espace $e = \Pi(\pi)$, dont les états correspondent aux instanciations de toutes les variables qui ne sont pas dans e. Ainsi, le nombre maximum de feuilles non nulles dans les arbres de probabilité agrégée est égal au nombre d'états engendrés par toutes les variables qui ne sont pas dans e.

Donnons la complexité de l'algorithme 14 dans le pire des cas, c'est-à-dire lorsque

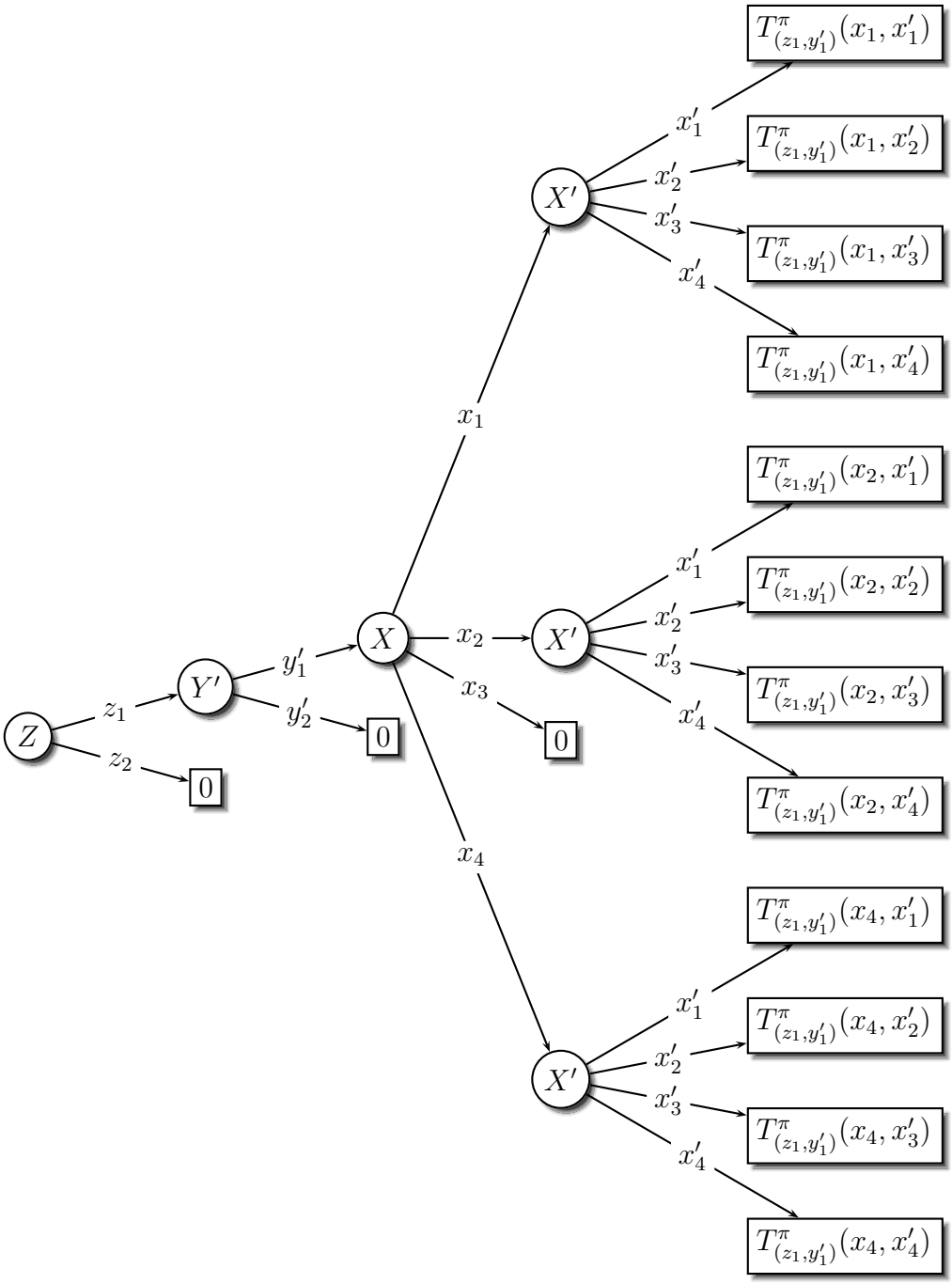


FIG. 4.3 – Arbre de probabilité des transitions locales d'une politique locale π du sous-MDP énuméré $\langle V_X, \mathcal{A}, \tilde{\mathcal{T}}_{(z_1,y'_1)}, \tilde{\mathcal{R}}_{(z_1,y'_1)} \rangle$ (cf. figure 3.7), définie dans la région $\tilde{x}_1 = \{x_1, x_2, x_4\}$

Algorithme 14 : Fonction `CalculeArbreProbabiliteAutresVariables`

Données : bdd_e (BDD), T^{π} (transitions factorisées agrégées), $addT_e^{\pi}$ (ADD), i, p
Résultat : $\tilde{T}_i^{\pi}$ (ADD)
début
  $add\_temp \leftarrow$ `addApply` $(\texttt{addTimes}, T^{\pi}[i], T_e^{\pi})$;
  $add\_num \leftarrow$ `ReadZero`();
  pour $\tilde{v}'_p \in \tilde{V}_p$ **faire**
    $cube\_vpp \leftarrow$ `addCube` $\left(\bigwedge_{X'^k_p \in \tilde{v}'_p} \left\{X'^k_p\right\}\right)$;
    $add\_temp\_bis \leftarrow$ `addExistAbstract`$(add\_temp, cube\_vpp)$;
    $add\_num \leftarrow$ `addApply`$(\texttt{addPlus}, add\_num, add\_temp\_bis)$;
  $cube\_vip \leftarrow$ `addCube` $\left(\bigwedge_{X'^k_i \in V_i} \left\{X'^k_i\right\}\right)$;
  $add\_denom \leftarrow$ `addExistAbstract`$(add\_num, cube\_vip)$;
  $add\_temp \leftarrow$ `addApply`$(\texttt{addDivide}, add\_num, add\_denom)$;
  $add\_temp\_bis \leftarrow$ `ReadZero`();
  pour $\tilde{v}_p \in \tilde{V}_p$ **faire**
    $cube\_vp \leftarrow$ `addCube` $\left(\bigwedge_{X^k_p \in \tilde{v}_p} \left\{X^k_p\right\}\right)$;
    $add\_temp\_ter \leftarrow$ `addExistAbstract`$(add\_temp, cube\_vp)$;
    $add\_temp\_ter \leftarrow$ `addApply` $\left(\texttt{addTimes}, add\_temp\_ter, \frac{1}{|\tilde{v}_p|}\right)$;
    $add\_temp\_bis \leftarrow$ `addApply`$(\texttt{addPlus}, add\_temp\_ter, add\_temp\_bis)$;
  $add\_temp \leftarrow$ `addApply`$(\texttt{addTimes}, add\_temp\_bis, \texttt{BddToAdd}(bdd\_e))$;
  retourner *add_temp;*
fin

toutes les états et toutes les transitions sont énumérés dans les ADD :

$$O\left(|\tilde{v}_p|^2 |V_i|\right)$$

A titre d'exemple, calculons un des arbres de probabilité de la variable Z, pour une politique π définie dans la région $\tilde{x}_1 = \{x_1, x_2, x_4\}$ et correspondant à l'instanciation $e = (z_1, y'_1)$ (cf. figure 3.7). D'après le théorème 9 et l'algorithme 14, cet arbre est obtenu à partir de l'arbre de probabilité agrégée $T^{\pi}[Z]$ (représenté dans la figure 4.2) et de l'arbre des transitions locales $addT^{\pi}_{(z_1,y'_1)}$ (représenté sous forme d'arbre dans la figure 4.3). Les différentes étapes du calcul sont :

1. Multiplication des arbres $T^{\pi}[Z]$ et $T^{\pi}_{(z_1,y'_1)}$. L'arbre A_1 obtenu est représenté sur la figure 4.4
2. Sommation de tous les sous-arbres de la variable X' dans l'arbre A_1. L'arbre A_2 obtenu est représenté sur la figure 4.5

3. Sommation de tous les sous-arbres de la variable Z' dans l'arbre A_2. L'arbre A_3 obtenu est représenté sur la figure 4.6
4. Division des arbres A_2 et A_3 puis sommation de tous les sous-arbres de la variable X dans le sous-arbre résultant de la division. L'arbre obtenu, divisé par la constante $|\tilde{x}_1| = 3$, est représenté dans la figure 4.7

4.5.4 Fonction `CalculeArbreRecompense`

D'après le théorème 9 :

$$\forall \left(\pi, \bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p\right) \in \tilde{\mathcal{A}} \times \left(\bigotimes_{j\neq p} V_j\right)^2 \times \tilde{V}_p \ , \ \forall\, \tilde{v}'_p \in \tilde{V}_p \ :$$

$$\tilde{R}^{\pi}\left(\bigwedge_{j\neq p} v_j, \bigwedge_{j\neq p} v'_j, \tilde{v}_p, \tilde{v}'_p\right) = \begin{cases} 0 & \text{si } \bigwedge\limits_{j\neq p} v_j \bigwedge\limits_{j\neq p} v'_j \notin \Pi(\pi) \\ \tilde{R}^{\pi}_{\bigwedge\limits_{j\neq p} v_j \bigwedge\limits_{j\neq p} v'_j}\left(\tilde{v}_p, \tilde{v}'_p\right) & \text{sinon} \end{cases}$$

L'arbre de récompense a donc une structure identique à celle de l'arbre de probabilité de $\tilde{X}_p$ (cf. fonction `CalculeArbreProbabiliteVariableReduite`). Le seul sous-arbre non nul est celui dont le chemin depuis la racine correspond à e. Ce sous-arbre a deux nœuds : les variables $\tilde{X}_p$ et $\tilde{X}'_p$. En réalité, seul un sous-arbre de la variable $\tilde{X}_p$ est non nul puisque la politique π n'est définie que dans une seule région du partitionnement $\tilde{V}_p$ (une seule valeur de la variable $\tilde{X}_p$). La fonction `CalculeArbreRecompense` est donnée dans l'algorithme 15.

Comme pour l'algorithme 11, la complexité de l'algorithme 15 est, si $|\tilde{V}_p|$ est grand : $O\left(dC|\tilde{V}_p|^2\right)$.

A titre d'exemple, considérons une politique locale π obtenue en décomposant le sous-MDP énuméré $\langle V_X, \mathcal{A}, \tilde{\mathcal{T}}_{(z_1,y'_1)}, \tilde{\mathcal{R}}_{(z_1,y'_1)}\rangle$, **définie dans la région** $\tilde{x}_1$. L'instanciation correspondant à cette politique est donc $e = \Pi(\pi) = (z_1, y'_1)$. L'arbre de récompense de la politique π est donné dans la figure 4.8. Ainsi que cela a été mentionné plus haut, l'arbre de récompense obtenu a la même structure que l'arbre de probabilité de la variable $\tilde{X}_p$, pour la même politique π et la même instanciation e (cf. figure 4.1).

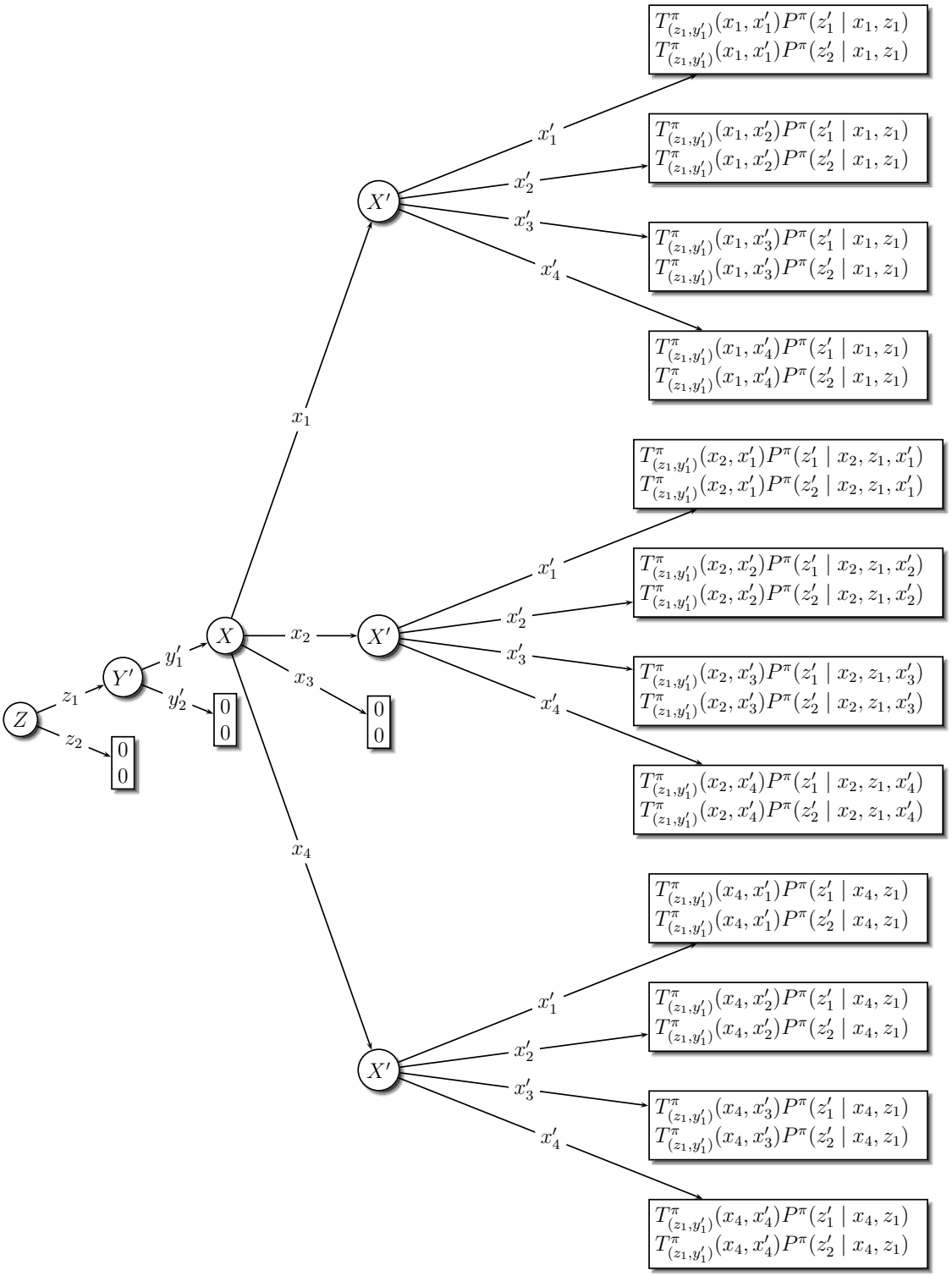


FIG. 4.4 – Arbre de décision A_1 obtenu en multipliant les arbres $T^{\pi}[Z]$ (cf. figure 4.2) et $T_{(z_1,y'_1)}(\pi)$ (cf. figure 4.3)

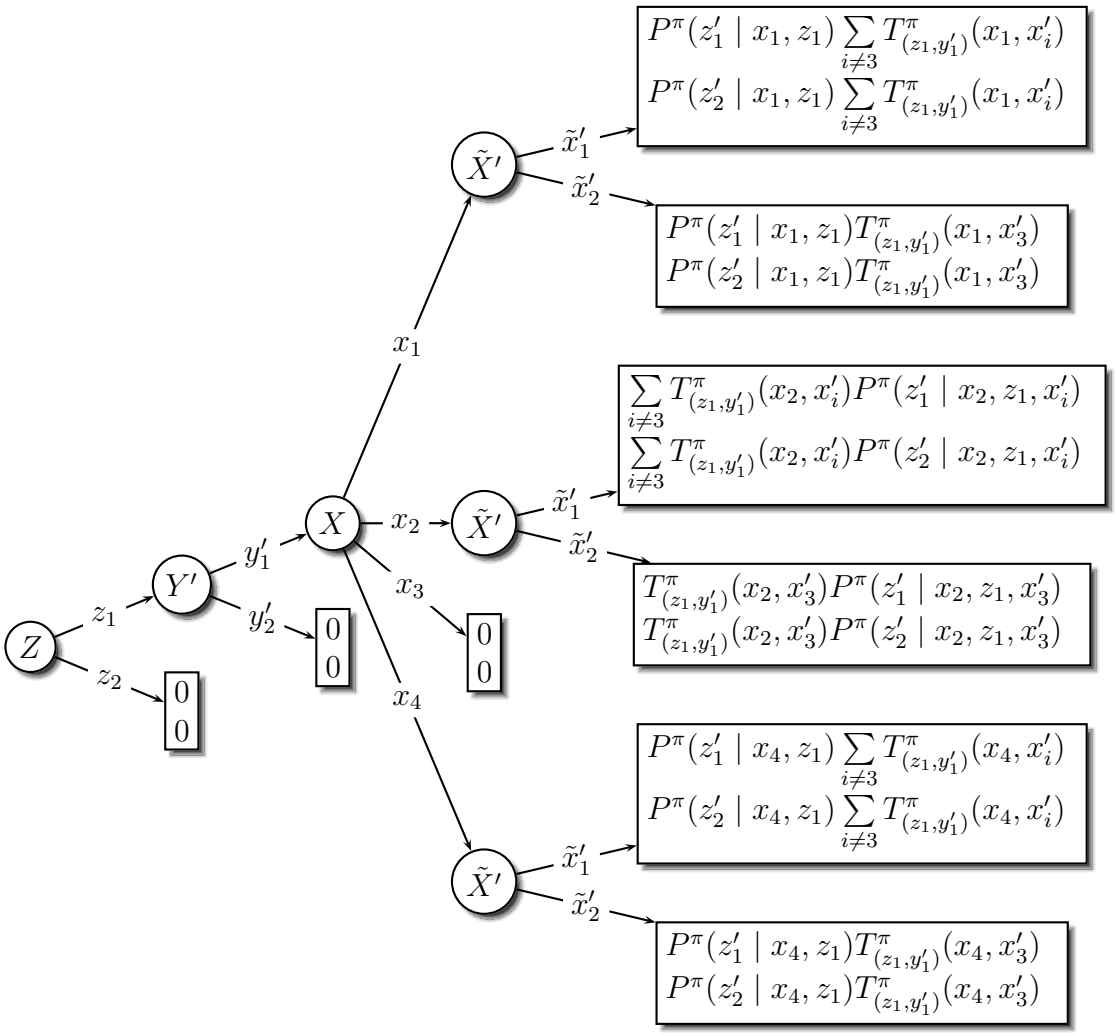


FIG. 4.5 – Arbre de décision A_2 obtenu en sommant tous les sous-arbres de la variable X' dans l'arbre de la figure 4.4

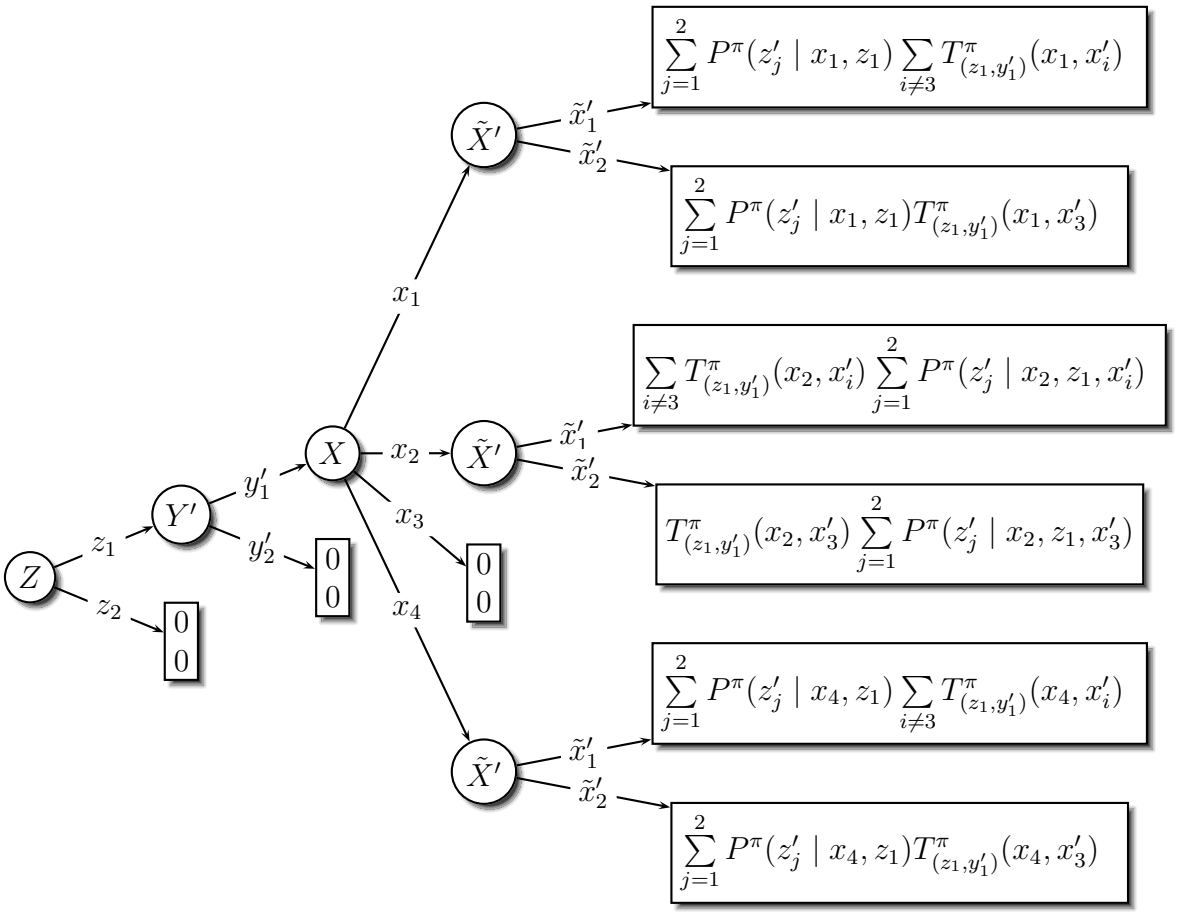
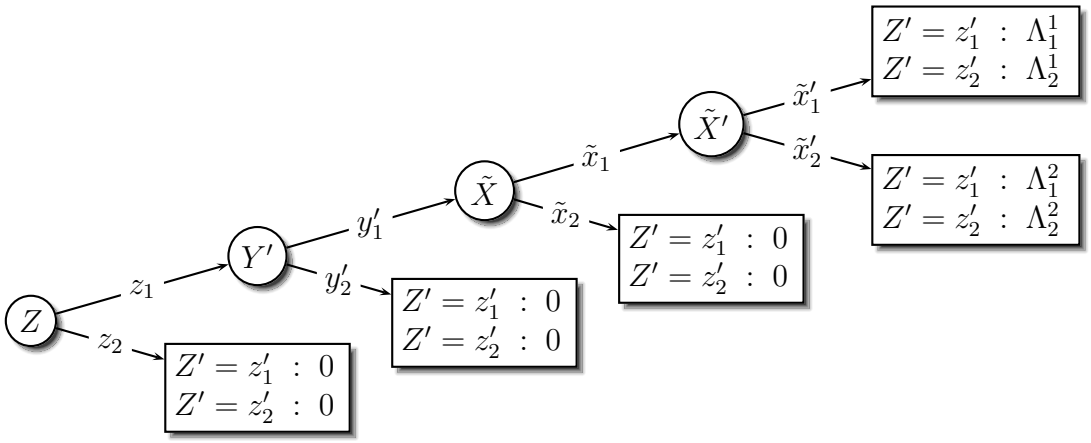


FIG. 4.6 – Arbre de décision A_3 obtenu en sommant tous les sous-arbres de la variable Z' dans l'arbre A_2 (cf. figure 4.5)



$$\Lambda^1_k = \frac{1}{3}\left(\frac{P^\pi(z'_k \mid x_1, z_1)\sum\limits_{i\neq 3} T^\pi_{(z_1,y'_1)}(x_1,x'_i)}{\sum\limits_{j=1}^{2} P^\pi(z'_j \mid x_1, z_1)\sum\limits_{i\neq 3} T^\pi_{(z_1,y'_1)}(x_1,x'_i)} + \frac{\sum\limits_{i\neq 3} T^\pi_{(z_1,y'_1)}(x_2,x'_i)P^\pi(z'_k \mid x_2, z_1, x'_i)}{\sum\limits_{i\neq 3} T^\pi_{(z_1,y'_1)}(x_2,x'_i)\sum\limits_{j=1}^{2} P^\pi(z'_j \mid x_2, z_1, x'_i)} + \frac{P^\pi(z'_k \mid x_4, z_1)\sum\limits_{i\neq 3} T^\pi_{(z_1,y'_1)}(x_4,x'_i)}{\sum\limits_{j=1}^{2} P^\pi(z'_j \mid x_4, z_1)\sum\limits_{i\neq 3} T^\pi_{(z_1,y'_1)}(x_4,x'_i)}\right)$$

$$\Lambda^2_k = \frac{1}{3}\left(\frac{P^\pi(z'_k \mid x_1, z_1)T^\pi_{(z_1,y'_1)}(x_1,x'_3)}{\sum\limits_{j=1}^{2} P^\pi(z'_j \mid x_1, z_1)T^\pi_{(z_1,y'_1)}(x_1,x'_3)} + \frac{T^\pi_{(z_1,y'_1)}(x_2,x'_3)P^\pi(z'_k \mid x_2, z_1, x'_3)}{T^\pi_{(z_1,y'_1)}(x_2,x'_3)\sum\limits_{j=1}^{2} P^\pi(z'_j \mid x_2, z_1, x'_3)} + \frac{P^\pi(z'_k \mid x_4, z_1)T^\pi_{(z_1,y'_1)}(x_4,x'_3)}{\sum\limits_{j=1}^{2} P^\pi(z'_j \mid x_4, z_1)T^\pi_{(z_1,y'_1)}(x_4,x'_3)}\right)$$

FIG. 4.7 – Arbre de probabilité de la variable Z du MDP factorisé abstrait $\langle \tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{\mathcal{T}}, \tilde{\mathcal{R}} \rangle$ pour une politique π définie pour l'instanciation $e = (z_1, y'_1)$ et dans la région $\tilde{x}_1 = \{x_1, x_2, x_4\}$

Algorithme 15 : Fonction `CalculeArbreRecompense`

Données : e (liste d'instanciation des variables de $\mathcal{K}_p$), $\tilde{R}_e^\pi$ (liste de récompenses énumérées), p
Résultat : $\tilde{R}^\pi$ (arbre de récompense)
début
  $\tilde{R}^\pi.racine() \leftarrow e.premier()$;
  $noeud \leftarrow \tilde{R}^\pi.racine()$;
  pour $v \in e$ **faire**
    pour $vv \in v.variable()$ **faire**
      si $vv = v.valeur()$ **alors**
        si $v = e.dernier()$ **alors**
          $noeud.fils(vv) \leftarrow \tilde{X}_p$;
          $noeud \leftarrow noeud.fils(vv)$;
          pour $\tilde{v}_p \in \tilde{V}_p$ **faire**
            $noeud.fils(\tilde{v}_p) \leftarrow \tilde{X}'_p$;
            $noeud \leftarrow noeud.fils(\tilde{v}_p)$;
            pour $\tilde{v}'_p \in \tilde{V}_p$ **faire**
              $noeud.fils(\tilde{v}'_p) \leftarrow feuille\left(\tilde{R}_e^\pi(\tilde{v}_p, \tilde{v}'_p)\right)$;
        sinon
          $noeud.fils(vv) \leftarrow e.suivant(v)$;
          $noeud \leftarrow noeud.fils(vv)$;
      sinon
        $noeud.fils(vv) \leftarrow feuille(0)$;
  retourner $\tilde{R}^\pi$;
fin

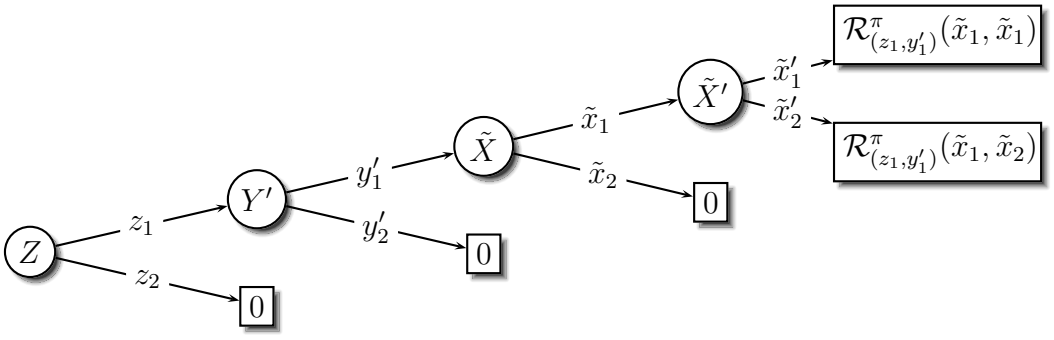


FIG. 4.8 – Arbre de récompense pour une politique locale π du sous-MDP énuméré $\langle V_X, \mathcal{A}, \tilde{\mathcal{T}}_{(z_1,y'_1)}, \tilde{\mathcal{R}}_{(z_1,y'_1)}\rangle$ (cf. figure 3.7)

Bilan

4.6 Un algorithme de réduction automatique d'arité des variables d'état

Dans cette partie, nous avons proposé un algorithme permettant de réduire automatiquement l'arité d'une variable d'état d'un MDP factorisé. Cet algorithme repose sur une énumération partielle des états du MDP factorisé en un sous-MDP énuméré, qui est ensuite décomposé en un sous-MDP énuméré abstrait de plus petite taille. La transformation inverse permet alors de reconstituer un MDP factorisé abstrait de plus petite taille puisqu'une de ses variables a bénéficié d'une réduction d'arité.

Cependant, ce procédé suppose que les variables d'état varient peu par rapport à celle dont nous souhaitons réduire l'arité. Nous pensons que de nombreux problèmes concrets vérifient cette hypothèse, **en particulier les problèmes d'exploration de zones**. En effet, lorsque l'agent se déplace dans l'environnement, seule la variable de position change avec une probabilité quasiment de 1 à chaque pas de déplacement. Dans les autres cas, une probabilité de changement quasi nulle signifie que l'action a échoué ou que l'action consiste à rester où l'agent se trouve. Les autres variables varient également, mais avec une probabilité très faible. Par exemple, la variable d'autonomie (niveau d'énergie restante) varie très peu à moins que le système soit défaillant au point que l'autonomie baisse soudainement de manière dramatique. Il est possible de prévoir une action qui arrête la mission lorsqu'une telle défaillance survient, si bien que notre procédé reste applicable (le fonctionnement est normal durant tout le processus de décision, sauf éventuellement le dernier).

Dans le cas où une variable Y varie fortement par rapport à la variable X dont nous souhaitons réduire l'arité, il est possible d'appliquer notre algorithme **à condition que X ne dépende pas de Y**. En effet, l'énumération des sous-MDP par rapport à la variable X est raisonnable si *uniquement* les variables dont X dépend varient peu : les politiques locales générées lors de la décomposition des sous-MDP énumérés ont une chance de réussir (atteindre un objectif ou sortir d'une région) que si les variables dont dépend X ne changent pas fréquemment – ce qui obligerait l'agent à changer inopinément de politique locale. Ainsi, si X dépend de Y, toutes

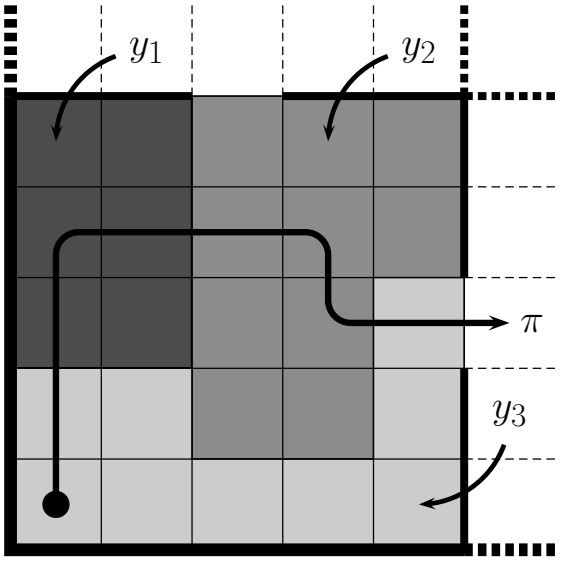


FIG. 4.9 – Exemple de problème de déplacement et de prise d'information pour lequel notre algorithme de réduction d'arité n'est pas approprié : la variable Y, dont dépend la variable de position X, varie au cours de l'application d'une politique locale qui vise à sortir d'une région, si bien que cette politique échoue puisque chaque politique locale est définie pour une valeur fixée des variables dont X dépend. Néanmoins, dans cette région, notre algorithme générera 3 classes de politiques locales pour chaque valeur de Y, si bien que la politique représentée réussira en tant que réunion de 3 politiques locales différentes.

les politiques locales risquent d'échouer au point de rendre la réalisation de la mission impossible ou très longue (le plan abstrait est très long puisqu'une nouvelle politique est nécessaire à chaque changement de valeur des variables autres que X). Par exemple, dans le cas d'un problème d'exploration, ce cas peut survenir si X est la variable de position et Y est une variable qui représente des zones de l'environnement différentes des régions topologiques résultant de la décomposition des sous-MDP énumérés par rapport à X (cf. figure 4.9).

4.7 Efficacité de l'approche choisie

La validation de notre approche nécessite de réaliser un grand nombre de tests sur des problèmes différents. Néanmoins, nous n'avons pas eu le temps de programmer cet algorithme, car nous l'avons conçu en toute fin de thèse. En dépit des tests qui devront être absolument réalisés dans le futur, nous présentons dans ce paragraphe quelques réflexions sur l'efficacité attendue de notre approche. Nous devrons aussi

comparer notre approche avec celles de [52, 54, 53] qui visent également à réduire la cardinalité de l'espace d'état, mais sans diminuer directement le nombre de variables du problème. Dans ces approches, l'espace d'états est hiérarchisé au cours de l'optimisation du MDP factorisé, sans l'énumérer localement. Elles sont donc orthogonales à notre algorithme `SVAR`, qui énumère et décompose localement le MDP préalablement à son optimisation.

Le but de notre algorithme est de réduire la cardinalité d'une variable d'état afin de diminuer la taille de l'espace d'états d'un MDP factorisé. Ceci représente un intérêt pour des variables d'état dont l'arité est très importante puisque la taille de l'espace d'état diminue dans la même proportion que l'arité de la variable réduite :

$$\frac{|\mathcal{V}|}{|\tilde{\mathcal{V}}|} = \frac{|V_p| \prod_{i \neq p} |V_i|}{|\tilde{V}_p| \prod_{i \neq p} |V_i|} = \frac{|V_p|}{|\tilde{V}_p|}$$

Ainsi, dans le cas d'un problème d'exploration dont la variable de position géographique passe d'une cardinalité très importante ($> 10^2$) à quelques régions ($\simeq 10$), la réduction de l'espace d'états est quasiment égale à la cardinalité initiale de la variable de position géographique. En termes de complexité, cela revient quasiment à enlever une variable d'état du modèle.

Dans le cas des BDD et des ADD, la complexité des calculs est proportionnelle à la taille des graphes représentant les données [68]. Il est donc important de calculer le nombre de variables binaires qui sont supprimées du modèle :

$$E_{\mathbb{N}}\left(\log_2(|V_p|)\right) + 1 - E_{\mathbb{N}}\left(\log_2(|\tilde{V}_p|)\right) - 1 \simeq E_{\mathbb{N}}\left(\log_2 \frac{|V_p|}{|\tilde{V}_p|}\right)$$

La taille maximale des ADD et des BDD étant une puissance de 2 du nombre de variables binaires, nous retrouvons évidemment la même réduction de complexité que précédemment. Dans le cas d'une variable de position géographique dont l'arité est de l'ordre de 10^k, la réduction du nombre de variables d'état diminue d'environ $k+1$, ce qui est avantageux puisque les algorithmes récents d'optimisation des MDP factorisés – mêmes heuristiques – se limitent à quelques dizaines de variables d'état. La réduction du nombre de variables binaires n'est certes pas très importante, mais elle peut s'avérer intéressante si notre approche est appliquée successivement à plusieurs variables d'état de grande arité. La réduction du nombre de variables peut alors être de l'ordre du nombre de variables auxquels l'algorithme est appliqué.

De plus, notre approche présente un intérêt non négligeable si les ADD sont utilisés à la place des arbres de décision. En effet, le temps d'accès aux données dans les ADD est en $O(\log N)$, où N est le nombre d'éléments non nuls stockés dans les ADD [69]. Or nous avons vu que les arbres abstraits de probabilité et de

récompense produits par notre algorithme ont de grande chance d'être très creux, certains ayant même qu'une seule feuille non nulle. Ainsi, indépendamment de la réduction de la taille de l'espace d'états, le temps d'accès aux données et donc le temps d'optimisation devraient être considérablement réduits.

Par conséquent, la complexité de l'optimisation du MDP factorisé abstrait a toutes les chances d'être considérablement réduite par rapport à celle du MDP factorisé initial. L'efficacité de notre approche repose donc entièrement sur la complexité de notre algorithme `SVAR` de réduction automatique d'arité des variables d'état. Sa complexité doit rester négligeable devant l'optimisation du MDP factorisé initial, mais elle est difficile à évaluer comme en témoignent les calculs de complexité que nous avons effectué. Nous avons montré que la majorité des fonctions de notre algorithme ont, dans le pire cas, une complexité polynomiale en la réduction d'arité de la variable réduite et le nombre de politiques locales générées. Seules deux fonctions (`PartitionneSousMDP` et `CalculeProbabilitésAgregees`) présentent une complexité exponentielle en :

- la cardinalité de la variable réduite $|\tilde{V}_p|$ (cf. équation 4.1 p. 99) ;
- la longueur d d'une instanciation de e, c'est à dire le nombre de variables dont dépend X_p et l'arbre de récompense, hormis X_p et X'_p (cf. équations 4.1 p. 99 et 4.2 p. 111).

Le premier cas est contrôlable via la fonction `CalculeDimensionPartitionnement` qui attribue une valeur à $|\tilde{V}_p|$. Nous n'avons pas actuellement de procédé automatique pour déterminer cette valeur. C'est donc une valeur d'entrée de l'algorithme, au même titre que le facteur d'actualisation γ ou la tolérance ϵ. Au vu de ce que nous avons dit plus haut dans ce paragraphe, il est raisonnable que l'arité de la variable réduite soit très faible. Dans le cas d'un problème d'exploration, on peut envisager un algorithme qui identifie automatiquement le *nombre* de régions topologiques de l'environnement, ce que ne fait pas l'algorithme de [25]. Dans les cas concrets (projet RESSAC par exemple), le nombre de régions géographiques est très faible, de l'ordre de 10. En tous les cas, il est primordial de remarquer que le nombre de régions du partitionnement ne dépend que de la topologie de ce dernier et non du nombre d'états énumérés, ce qui permet de **borner le terme exponentiel en la cardinalité de la variable réduite indépendamment de la taille des données**.

Quant à d, ce nombre peut être important car l'arbre de récompense peut dépendre d'un grand nombre de variables d'état. Dans le cas d'un problème d'exploration, nous avons vu que d est de l'ordre du nombre d'objectifs d'exploration à atteindre. Or les objectifs sont en général de l'ordre du nombre de régions géographiques, si bien que nous pouvons également borner le terme exponentiel en d indépendamment de la taille des données. Dans le cas général, l'équation de complexité 4.2 (p. 111) montre que le terme exponentiel en d est multiplié par le nombre d'itérations nécessaires pour que les probabilités agrégées convergent. Intuitivement,

il est raisonnable de penser que les probabilités agrégées convergent d'autant plus vite que l'hypothèse de découplage est vérifiée, puisque les probabilités de transition font très peu évoluer les probabilités agrégées d'un instant à l'autre du processus de mise à jour des probabilités agrégées. Ainsi, même si d est grand, le nombre d'itérations sur les probabilités agrégées devrait être faible à condition que l'hypothèse de découplage soit vérifiée, ce qui limite le nombre de calculs de complexité exponentielle en d.

Néanmoins, seuls de nombreux tests de notre algorithme sur des problèmes différents permettraient d'évaluer notre approche. Précisons tout de même qu'il est certain que notre algorithme permet d'économiser de la mémoire physique puisqu'il réduit la taille de l'espace d'états. Même dans le cas où son temps d'exécution serait très important, il aurait quand même un avantage sur l'optimisation directe du MDP factorisé initial, qui ne pourrait même pas être effectuée sur des problèmes de très grande taille, si la mémoire physique est limitée. Ainsi, si l'optimisation du MDP est primordiale au détriment du temps de calcul, notre approche «diviser pour régner» présente dans tous les cas l'avantage de réduire la complexité de l'optimisation du MDP. En effet, notre algorithme n'est pas plus complexe que l'optimisation directe du MDP initial : la seule fonction comparable à l'optimisation directe est la fonction `CalculeProbabilitesAgregees` (principe identique à l'équation de Bellman), qui a toutes les chances d'être efficace si d est petit et si l'hypothèse de découplage est vérifiée.

4.8 Problème non résolu : modélisation du MDP factorisé initial

Notre algorithme `SVAR` repose sur un modèle factorisé dont une variable au moins a un grand nombre valeurs possibles. Cependant, à moins que le modèle soit automatiquement généré, la modélisation des MDP factorisés est fastidieuse – voire impossible – lorsque certaines variables ont un grand nombre valeurs. Les arbres de probabilité et de récompense sont alors très larges au point qu'ils deviennent rapidement illisibles. En particulier, pour un problème d'exploration dont la variable de position géographique a généralement entre une centaine et un millier de valeurs, la définition manuelle du modèle factorisé de la mission est rédhibitoire.

Ainsi, dans la partie suivante, nous proposons une méthode permettant de définir le MDP factorisé à un niveau macroscopique, avec des variables de faible arité uniquement. Les actions seront des macro-actions *a priori inconnues*, permettant de transiter entre les différentes valeurs d'une macro-variable donnée. Par exemple, le modèle factorisé d'une mission d'exploration sera défini en fonction d'une variable de région géographique (au lieu de position géographique microscopique) ; les ac-

tions seront les politiques locales définies dans ces régions, inconnues au moment de la modélisation. Le modèle factorisé sera donc paramétré par ces politiques locales inconnues, qui pourront être générées par décomposition du sous-espace de navigation.

La différence majeure de cette approche avec celle que nous présenté dans cette partie repose sur la donnée, par l'utilisateur, du partitionnement d'une variable de grande arité et du modèle directement abstrait. La réduction d'arité n'est donc pas automatique, elle est fournie par l'utilisateur dans le modèle, en formulant des hypothèses sur les politiques locales qui résultent de cette réduction d'arité.

Troisième partie

Réseaux Bayésiens Dynamiques Génériques et Hiérarchiques pour l'aide à la modélisation de MDP factorisés

Motivation

4.9 Intérêt des modèles factorisés de MDP

Les MDP factorisés [26, 31] sont une représentation efficace des problèmes de décision dans l'incertain dont l'espace d'états est factorisé en différentes composantes. Un grand nombre de problèmes concrets sont naturellement factorisés, au point qu'il est plus facile de modéliser le problème directement à l'aide de variables d'état et de transitions stochastiques entre ces différentes variables, en utilisant des Réseaux Bayésiens Dynamiques (DBN, cf. section 2.2 page 43). Un problème de déplacement et de prise d'information, par exemple, est défini au moins par des variables de position géographique, d'objectifs représentant les zones explorées ou à explorer, d'autonomie (niveau restant d'énergie). D'autres variables telles que les conditions météorologiques, l'altitude pour un drone ou encore la qualité des communications radio peuvent être prises également en compte.

Alors qu'il est facile, dans un MDP factorisé, de rajouter des variables supplémentaires et leurs transitions **sans modifier les variables et les transitions existantes**, ceci est impossible dans un MDP énuméré sans reformuler tout l'espace d'états et, par conséquent, toutes les transitions (les états énumérés sont des conjonctions des valeurs des variables d'état). Les MDP factorisés offrent donc plus de souplesse de modélisation que leurs homologues énumérés. De plus, les MDP factorisés permettent de modéliser des problèmes de grande taille qui ne peuvent l'être avec des MDP énumérés à cause de la mémoire physique considérable qu'ils requièrent (cf. [26, 16, 5, 21]).

4.10 Limite des MDP factorisés

Pour autant, les transitions des MDP factorisés sont représentées par des arbres de transition qui sont difficiles à modéliser lorsqu'une variable d'état a un grand nombre de valeurs possibles. Les arbres de transition sont alors très larges, et leur modélisation devient rapidement rédhibitoire lorsque la profondeur des arbres augmente. La figure 4.10 montre un exemple d'arbre de transition de la variable de

position géographique pour un problème d'exploration. Cet exemple étant de très petite taille, il est facile d'imaginer la difficulté de modéliser – avant même de résoudre – un problème concret ayant plusieurs centaines d'états géographiques. Si la topologie d'un sous-espace d'états est connue, il est parfois possible de modéliser ce sous-espace à l'aide de plusieurs variables d'état, ce qui structure un peu plus et facilite donc la modélisation des transitions. Par exemple le sous-espace de navigation dans un problème d'exploration peut être modélisé avantageusement avec les 3 variables spatiales (x, y, z), mais la modélisation dans ce cas reste difficile (cf. figure 4.11)[1].

Même si cette variable est convertie en variables binaires, la modélisation reste difficile car ces variables binaires n'ont pas de signification directe par rapport à la variable initiale. Considérons par exemple une variable de position géographique prenant 8 valeurs possibles. Une représentation binaire naïve consiste à transformer cette variable en 8 variables binaires, chacune représentant une valeur possible de la variable initiale. Cette représentation n'est pas efficace pour deux raisons : non seulement elle augmente inutilement la taille de l'espace d'états ($2^8 - 8 = 148$ états supplémentaires sont générés !) mais elle nécessite d'imposer des relations qui indiquent que 2 valeurs "**true**" simultanées pour 2 de ces variables est "**false**" (il est impossible que la variable initiale puisse avoir simultanément 2 valeurs différentes).

Une meilleure représentation binaire de la variable initiale, non redondante, consiste à générer autant de variables binaires que le logarithme en base 2 de l'arité de la variable initiale, de sorte que la transformation soit injective (elle n'est pas surjective car le logarithme en base 2 n'est pas nécessairement un entier). Dans le cas de notre variable de position géographique d'arité 8, nous générerions donc 3 variables binaires. Malheureusement, cette transformation minimale nécessite de garder en mémoire la correspondance entre les valeurs des variables binaires générées et les valeurs de la variable initiale puisque, contrairement à la représentation naïve, chaque variable binaire ne correspond pas à une valeur particulière de la variable d'état initiale.

4.11 Modélisation hiérarchique des MDP factorisés

Nous proposons, lorsque c'est possible, de tirer profit de la topologie du sous-espace engendré par une variable d'état afin de faciliter la modélisation des transitions factorisées où elle est impliquée. Reprenons par exemple le MDP représenté dans les figures 4.10 et 4.11. Le sous-espace de la variable de navigation est constitué

[1] Il est possible de vérifier que les MDP factorisés des figures 4.10 et 4.11 sont équivalents grâce aux théorèmes 7 (p. 66) et 8 (p. 68).

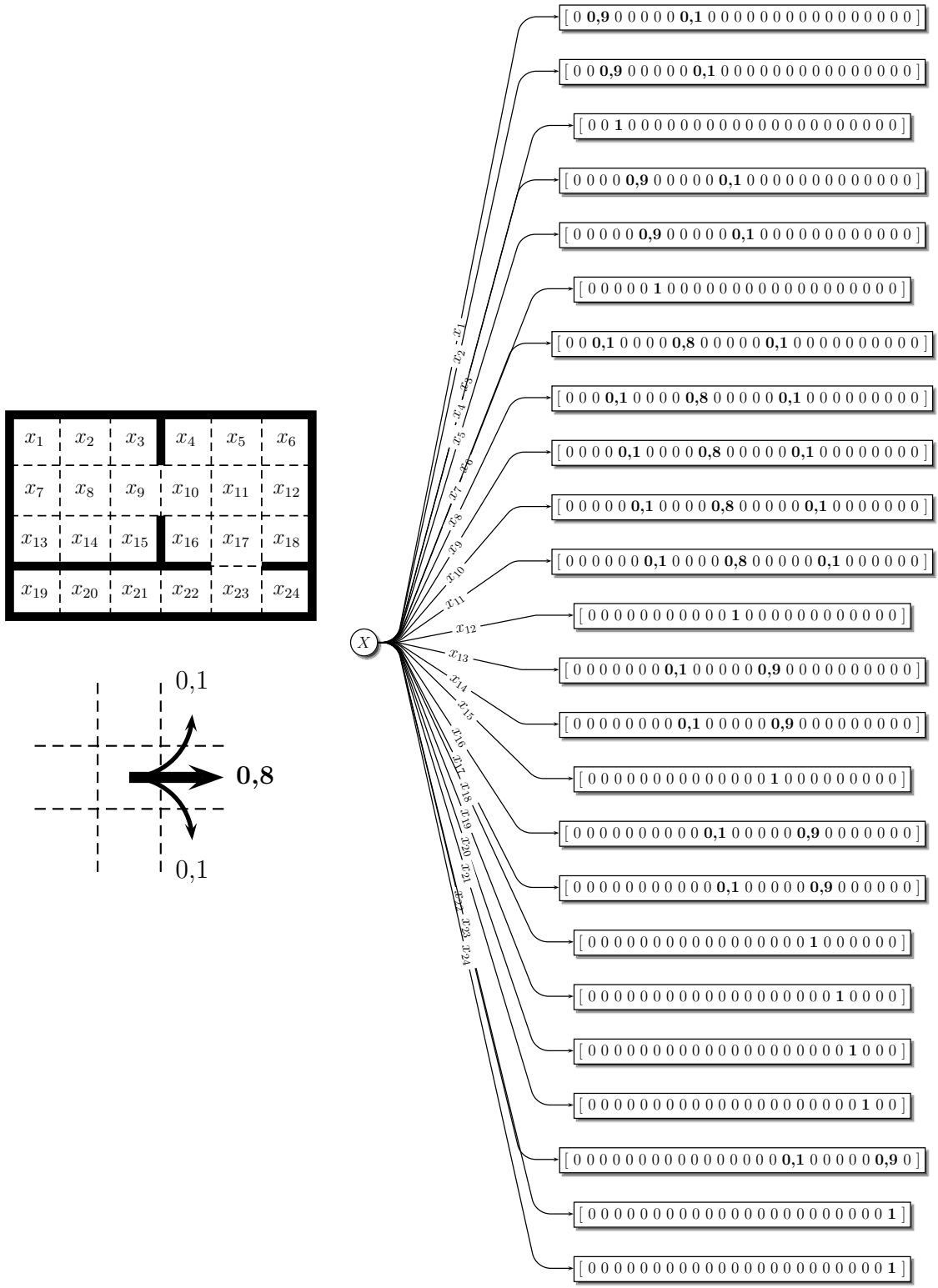


FIG. 4.10 – Exemple d'arbre de transition pour l'action «aller à droite» de la variable X de position géographique pour un problème d'exploration de petite taille.

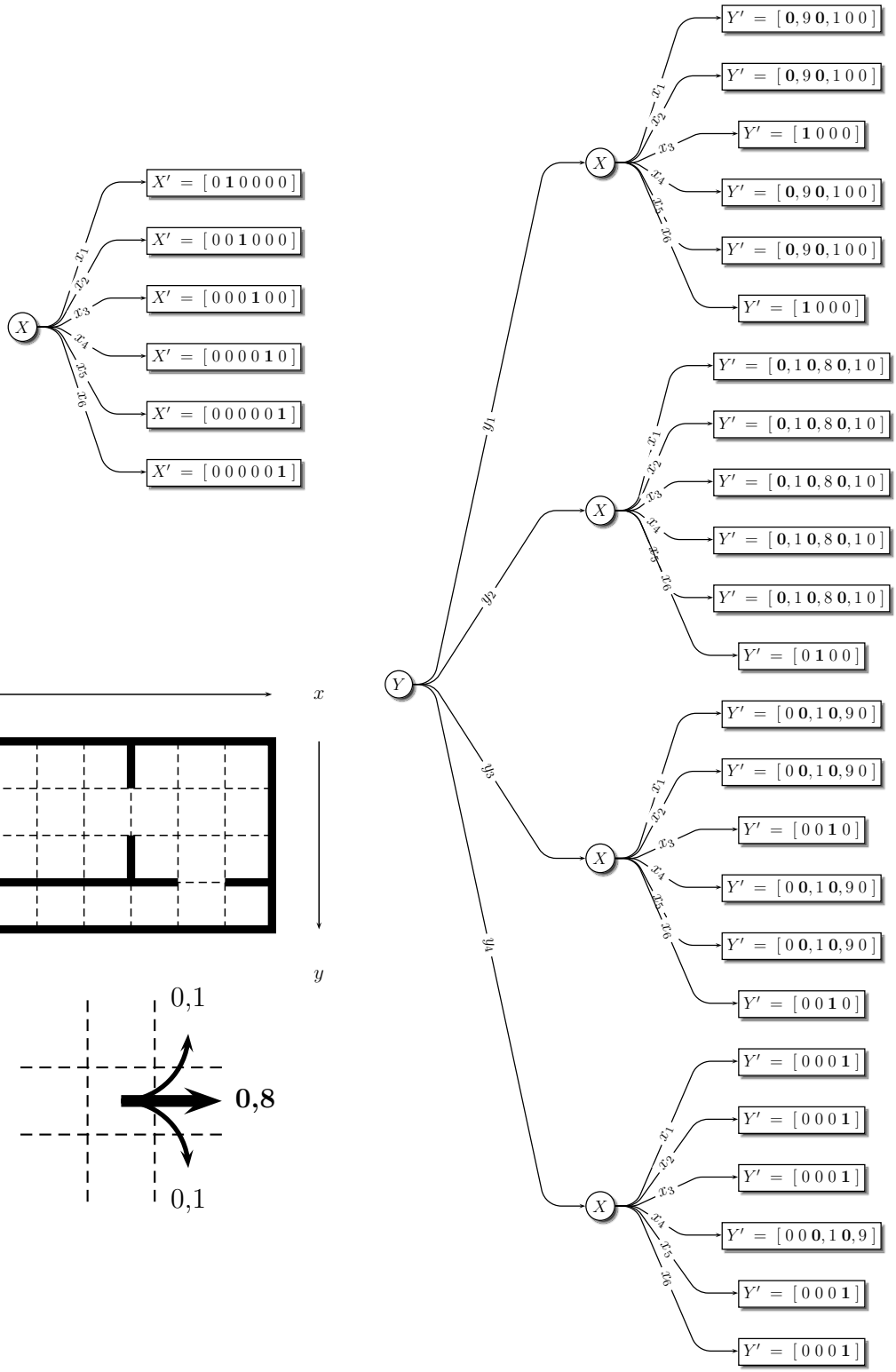


FIG. 4.11 – Exemple d'arbre de transition pour l'action «aller à droite» des variables X et Y de position géographique pour un problème d'exploration de petite taille.

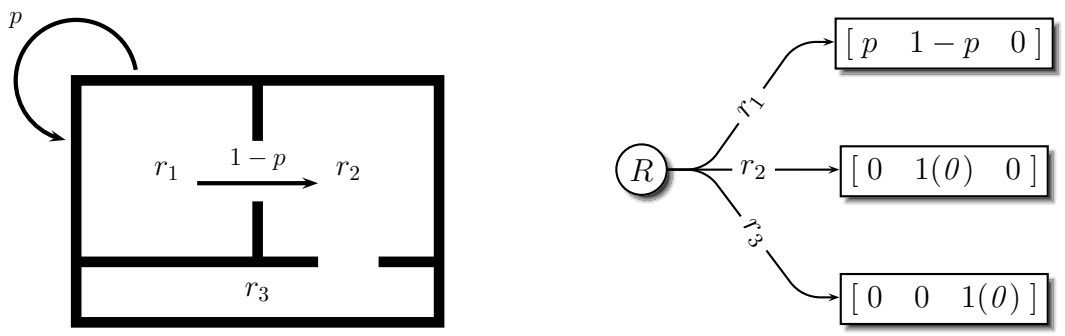


FIG. 4.12 – Exemple d'arbre de transition de la variable abstraite R pour une politique locale π définie dans la région r_1. Les feuilles de r_2 et r_3 ne sont pas nécessairement des lois de probabilité car la politique locale n'est pas définie dans ces régions.

de 3 régions si bien qu'il es possible de considérer les transitions de cette variable au niveau plus abstrait des régions. La variable abstraite correspondante n'a plus que 3 valeurs possibles, ce qui simplifie grandement la modélisation des arbres de transition du MDP factorisé.

Néanmoins, les actions du MDP initial ne s'appliquent plus sur la variable d'état abstraite «région». Pour passer d'une région à une autre, plusieurs actions sont nécessaires : seules des politiques locales définies dans chaque région permettent de transiter entre les différentes régions de l'environnement. **Les actions du MDP factorisé seront donc des politiques locales définies dans les régions du sous-espace engendré par la variable abstraite.** La figure 4.12 montre l'arbre de transition de la variable abstraite «région» pour une politique locale donnée. Cet arbre est beaucoup plus simple et concis que ceux des figures 4.10 et 4.11 alors qu'ils représentent tous trois *le même problème de décision.*

Cependant, il existe un grand nombre de politiques locales possibles dans chaque région ($|\mathcal{E}_r|^{|\mathcal{A}|}$ pour une région $\mathcal{E}_r$ et des actions $\mathcal{A}$). Il n'est donc pas raisonnable de définir autant de DBN que de politiques locales possibles : chaque arbre de transition est certes plus simple à modéliser, mais beaucoup plus d'arbres doivent être définis !

4.12 Méthodologie

Afin de réduire le nombre de DBN abstraits à modéliser, nous proposons d'utiliser deux techniques distinctes, la deuxième étant une contribution propre à notre modèle factorisé hiérarchique :

- décomposition (génération d'un sous-ensemble suffisant de politiques locales) du sous-espace engendré par la variable abstraite
- définition d'un seul DBN abstrait générique paramétré par les politiques locales

et les régions de la décomposition

Les techniques de décomposition (cf. section 2.1 page 29) permettent de calculer un *ensemble exhaustif mais minimal* de politiques locales, de sorte que le déplacement microscopique à l'intérieur de chaque région soit optimal au regard du reste de l'environnement. Le nombre de politiques locales générées reste important (cf. section 2.1 page 29) mais les régions sont généralement similaires par rapport aux autres variables d'état, en ce sens que les transitions des autres variables d'état dépendent des régions atteignables de toute politique locale mais pas de certaines régions particulières. Nous verrons qu'il est alors possible de définir un seul DBN générique paramétré par les politiques locales et les régions. Dans le cas où une variable d'état dépendrait de certaines régions, cela signifierait que le modèle est par essence défini au niveau macroscopique des régions, ce qui sort du cadre de la méthodologie que nous présentons dans ce chapitre.

Afin de résoudre le MDP factorisé, tous les DBN doivent être définis. Nous présenterons un algorithme que nous avons développé afin d'instancier automatiquement les DBN définis pour chaque politique locale de la variable abstraite. Cet algorithme analyse la syntaxe du DBN générique et en déduit les transitions qu'il doit générer pour chaque politique locale. Les DBN «concrets» sont donc générés automatiquement si bien que le modèle du MDP factorisé ne nécessite de définir qu'un seul DBN relativement simple, à condition de respecter une syntaxe particulière, que nous définissons dans la suite.

Plan de la partie III

Chapitre 5 : proposition de définition d'un Réseau Bayésien Dynamique Générique et Hiérarchique, et d'un d'algorithme d'instanciation automatique d'un tel DBN

Chapitre 6 : tests de notre approche générique et hiérarchique sur des missions de déplacement et de prise d'information du type ReSSAC

Chapitre 5

Syntaxe et algorithme d'instanciation automatique des Réseaux Bayésien Dynamiques Génériques et Hiérarchiques

5.1 Définition d'une sémantique pour un réseau bayésien dynamique et hiérarchique

5.1.1 Variable d'état abstraite

Le but de cette partie est de définir plus facilement les transitions où une variable d'état de grande arité est impliquée. Notons X_p une telle variable. Contrairement à l'algorithme `SVAR` (p. 94) qui calcule automatiquement le partitionnement optimal du sous-espace engendré par la variable X_p, nous supposons que la topologie de ce sous-espace est connu, au point que son partitionnement, noté $\tilde{X}_p$, est une donnée du modèle. Il peut être donné «manuellement» ou bien calculé automatiquement avec des algorithmes de partitionnement automatique de graphe [25, 73]. Dans le cas d'une mission de déplacement et de prise d'information de type ReSSAC, le sous-espace de navigation est défini en amont sous la forme de régions (montagnes, forêts, zones d'exploration, etc.) renfermant des points de passage. Le partitionnement est, dans ce cas particulier, donné dans le modèle comme pré-requis de la définition des points de passage.

Les actions permettant de transiter entre les différentes régions du partitionnement $\tilde{X}_p$ sont des politiques locales définies sur les valeurs microscopiques de la variable X_p (points de passage dans le cas du projet ReSSAC). Ces politiques locales peuvent être calculées automatiquement [23, 24, 22] de sorte que le déplacement

microscopique de l'agent dans chaque région soit optimal au regard des autres variables ou des autres régions. Il peut être aussi bien donné par l'opérateur en tant que pool de règles locales à appliquer pour prendre un objectif interne d'une région, ou pour aller d'une région à une autre. Cette dernière méthode ne garantie cependant pas l'optimalité du pool de politiques locales. En tous les cas, nous supposons qu'un ensemble de politiques locales sont connues, notées $\left(\Pi_p^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{X}_p}$:

$$\forall\ \tilde{v}_p \in \tilde{X}_p\ ,\ \forall\ \pi \in \Pi_p^{\tilde{v}_p}\ ,\ \pi\ :\ \tilde{v}_p \rightarrow \mathcal{A}$$

Le pool de politiques locales $\left(\Pi_p^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{X}_p}$ étant potentiellement très grand, nous proposons de définir un seul Réseau Bayésien Dynamique (DBN) générique **paramétré par une politique locale quelconque π du pool de politiques locales**. Chaque politique locale est définie dans une seule région si bien que le DBN générique est implicitement paramétré par la région où est définie la politique locale variable du DBN. Notons $\tau\ :\ \left(\Pi_p^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{X}_p} \rightarrow \tilde{X}_p$ la fonction qui à une politique locale fait correspondre la région où elle est définie.

Syntaxe de la variable réduite $\tilde{X}_p$ dans les arbres de décision du DBN générique

Soit π la politique paramétrable du DBN générique. Cette politique n'agit que sur la région $\tau(\pi)$ où elle est définie, si bien que les sous-arbres d'un nœud $\tilde{X}_p$ sont de deux types :

- sous-arbre correspondant à la valeur $\tau(\pi)$
- autres sous-arbres **nuls** correspondant aux valeurs différentes de $\tau(\pi)$

Les nœuds du second type étant toujours nuls, il est préférable de définir un masque sous forme d'arbre de décision binaire qui permet de restreindre l'application de π au sous-espace d'état $\tilde{X}_p = \tau(\pi)$. Les arbres de probabilité et les arbres de valeur (fonction de valeur et récompense) devront être systématiquement multipliés par cet «arbre de masque» durant l'optimisation du MDP. L'arbre de masque peut, bien entendu, contenir d'autres variables comme la variable d'autonomie : une politique n'est applicable que si l'agent a suffisamment d'autonomie pour l'appliquer. Il est important qu'un solveur de MDP factorisé prenne en compte les arbres de masque car ils permettent de restreindre l'application des actions à des sous-espaces d'états (contrairement à [26, 37]), ce qui contribue à améliorer l'efficacité des algorithmes d'optimisation des MDP factorisés.

Les nœuds de la variable $\tilde{X}_p$ n'ont donc que deux sous-arbres abstraits, un pour $\tau(\pi)$ et un autre pour le complémentaire $\overline{\tau(\pi)}$, en notant $\overline{\tau(\pi)}$ l'ensemble des valeurs

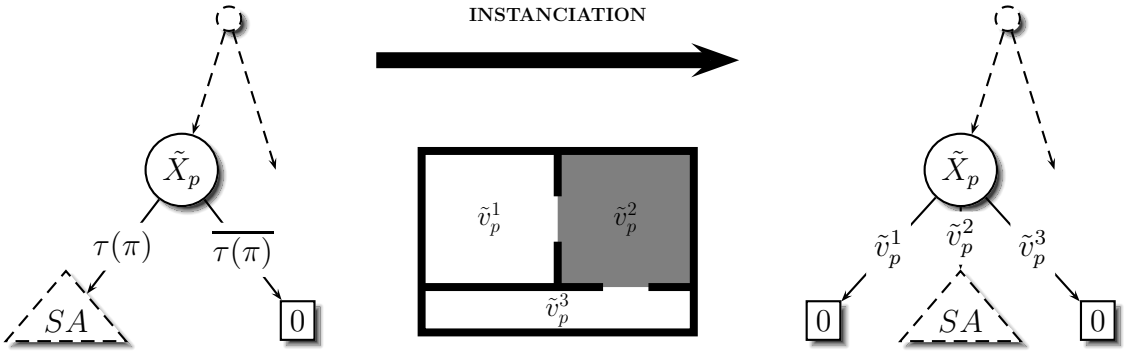


FIG. 5.1 – Exemple d'arbre de masque du DBN générique, ainsi qu'une de ses instanciations possibles pour la région $\tilde{v}_p^2$ d'un sous-espace de navigation constitué de 3 régions

de $\tilde{X}_p$ différentes de $\tau(\pi)$ (où la politique pi n'est pas définie) :

$$\overline{\tau(\pi)} = \left\{ \tilde{v}_p \in \tilde{X}_p \ : \ \tilde{v}_p \neq \tau(\pi) \right\} = \tilde{V}_p \setminus \{\tau(\pi)\}$$

La figure 5.1 représente un exemple d'arbre de masque du DBN générique, ainsi qu'une de ses instanciations possibles.

Syntaxe de la variable réduite $\tilde{X}'_p$ dans les arbres de décision du DBN générique

Dans le cas de la variable réduite $\tilde{X}'_p$, nous devons distinguer les valeurs atteignables par la politique π, et celles qui ne le sont pas. Notons $\zeta \ : \ \left(\Pi_p^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{X}_p} \rightarrow \tilde{X}_p$ la fonction qui à une politique locale π fait correspondre l'ensemble des valeurs de la variable réduite atteignables par π. Ainsi, π est une fonction de $\tau(\pi)$ dans $\zeta(\pi)$. Les sous-arbres abstraits d'un nœud $\tilde{X}'_p$ sont donc de deux types :

- sous-arbres correspondant à une valeur de $\zeta(\pi)$
- sous-arbres correspondant à une valeur de $\overline{\zeta(\pi)}$, où $\overline{\zeta(\pi)} = \tilde{V}_p \setminus \zeta(\pi)$ est l'ensemble des valeurs de $\tilde{X}_p$ qui ne sont pas atteignables par π

La figure 5.2 représente un exemple d'arbre de décision du DBN générique contenant la variable $\tilde{X}'_p$, ainsi qu'une de ses instanciations possibles.

5.1.2 Variables définies par rapport à la variable réduite

La modélisation du MDP factorisé sous forme abstraite permet, au besoin, d'introduire des variables d'état définies pour chaque valeur de la variable réduite. Considérons par exemple le cas d'un problème d'exploration pour lequel chaque région du sous-espace de navigation contient un seul objectif à atteindre. Les variables

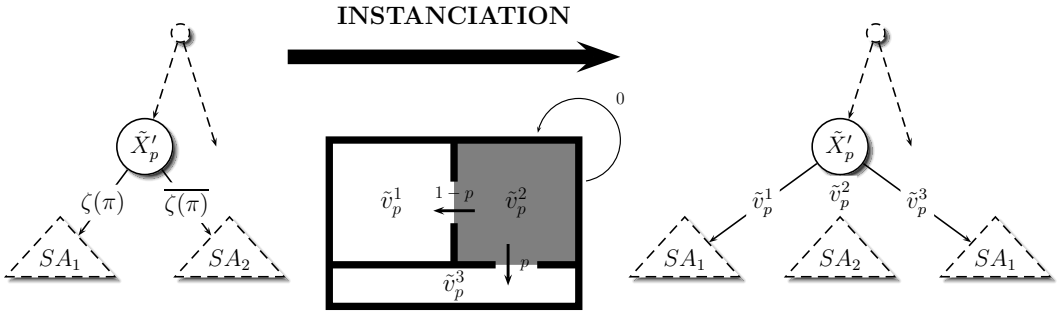


FIG. 5.2 – Exemple d'arbre de décision du DBN générique contenant la variable réduite $\tilde{X}'_p$, ainsi qu'une de ses instanciations possibles pour la région $\tilde{v}_p^2$ d'un sous-espace de navigation constitué de 3 régions

«objectif» sont donc définies pour chaque valeur de la variable abstraite de région géographique. Ces variables sont aussi nombreuses que le nombre de valeurs de la variable réduite.

Seule la variable définie pour la région où s'applique la politique locale paramètre du DBN générique est susceptible de changer. Ou, du moins, le changement des autres variables est identique. Dans le cas des variables d'état «objectif», seul l'objectif contenu dans la région où s'applique la politique locale peut être atteint, à moins que la politique locale consiste à sortir de la région car l'objectif qu'elle renferme est déjà atteint. Les autres objectifs, quant à eux, n'ont aucune chance d'être atteints car la politique locale n'agit pas sur les états de navigation où se trouvent ces objectifs.

Les variables définies par rapport à la variable réduite, et donc leurs arbres de probabilité, sont de deux types :

- variable définie dans la région $\tau(\pi)$ où s'applique la politique locale paramétrable
- variable abstraite définie dans les autres régions $\overline{\tau(\pi)}$, représentant toutes les variables définies dans $\overline{\tau(\pi)}$

La figure 5.3 représente un exemple d'arbres de probabilité d'une variable Y définie pour chaque valeur de la variable réduite $\tilde{X}_p$, ainsi qu'une de leurs instanciations possibles.

Il n'est pas autorisé qu'un arbre de décision contienne le nœud abstrait de la variable abstraite définie dans les autres régions $\overline{\tau(\pi)}$. En effet, ce nœud abstrait correspond à $|\tilde{V}_p| - 1$ nœuds après instanciation de chaque politique locale, ce qui n'est pas nécessairement égal à l'arité de la variable d'état contenue dans le nœud parent du nœud abstrait. Cette syntaxe serait possible uniquement dans le cas où l'arité de la variable d'état parent serait égal au moins à $|\tilde{V}_p| - 1$, mais il n'y aurait tout de même aucune raison de distinguer les différentes valeurs de cette variable

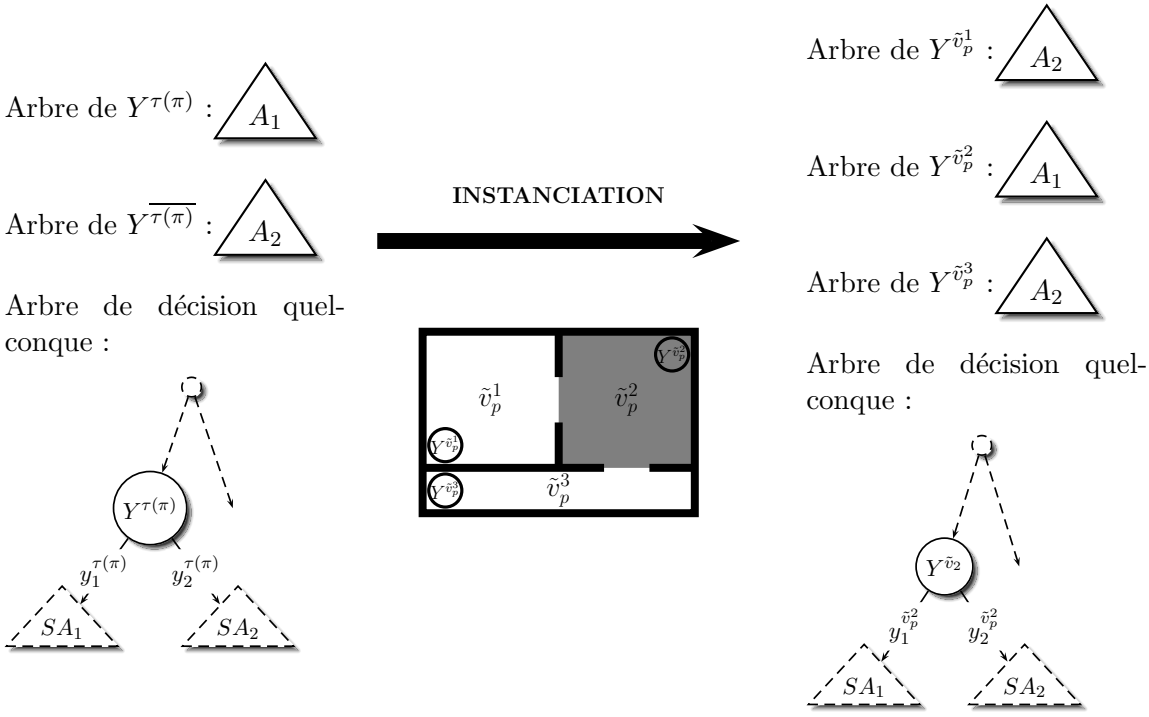


FIG. 5.3 – Exemple d'arbres de probabilité d'une variable Y définie pour chaque valeur de la variable réduite $\tilde{X}_p$, ainsi qu'une de leurs instanciations possibles pour la région $\tilde{v}_p^2$

par rapport à des régions où la politique ne s'applique pas (comment classer ses valeurs lors de l'instanciation automatique des politiques locales ?). En revanche, il est autorisé qu'un nœud de la variable définie dans la région $\tau(\pi)$ soit incluse dans un arbre de décision, puisqu'elle est unique (cf. figure 5.3).

5.1.3 Feuilles abstraites de l'arbre de probabilité de la variable réduite

Chaque feuille de l'arbre de probabilité de la variable réduite est une liste de probabilités, qui indique les probabilités d'obtenir chaque valeur de $\tilde{X}'_p$ connaissant l'instanciation des nœuds parents. La seule distinction possible entre les valeurs de $\tilde{X}'_p$ est au niveau de l'atteignabilité des régions par π. Les feuilles de l'arbre de probabilité de $\tilde{X}_p$ sont donc des listes à deux éléments abstraits :

- le premier élément correspond à la probabilité d'arriver dans une région atteignable quelconque,
- le deuxième élément, qui est la probabilité d'arriver dans une région non atteignable, est toujours nul.

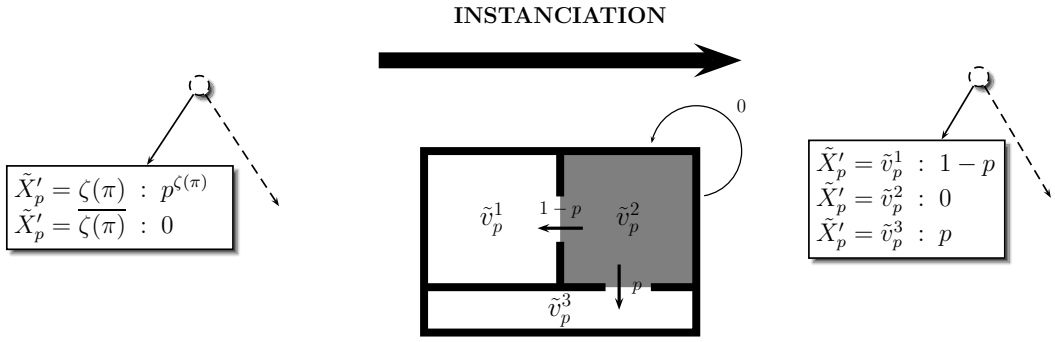


FIG. 5.4 – Exemple de feuille générique de l'arbre de probabilité de la variable réduite $\tilde{X}_p$, ainsi qu'une de ses instanciations possibles pour la région $\tilde{v}_p^2$ d'un sous-espace de navigation constitué de 3 régions

Notons $p^{\zeta(\pi)}$ la probabilité abstraite d'arriver dans une région quelconque de $\zeta(\pi)$. Lors de l'instanciation des probabilités d'obtenir chaque valeur atteignable $\tilde{v}'_p$ de $\tilde{X}'_p$, la probabilité abstraite $p^{\zeta(\pi)}$ sera remplacée par la probabilité d'obtenir $\tilde{v}'_p$ (voir figure 5.4). Cette probabilité est une donnée de notre modèle hiérarchique : à chaque politique locale π correspond une probabilité stationnaire $P^\pi : \tau(\pi) \times \zeta(\pi) \to [0;1]$, qui est le point fixe de la chaîne de Markov définie par les états de la région $\tau(\pi)$ et les transitions de la politique π (cf. section 2.1 page 29 et [23, 24, 57]).

5.1.4 Paramétrage des probabilités de transition

Les probabilités d'obtenir les différentes values de certaines variables d'état peut dépendre des probabilités de transition entre les régions de $\tilde{V}_p$. Considérons par exemple les variables «objectif» $\left(Y^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{V}_p}$ de la figure 5.3. Si un objectif n'est pas encore atteint, la probabilité de l'atteindre avec une politique visant à obtenir cet objectif dépend de la probabilité de rester dans la région qui contient cet objectif. Autrement dit, la probabilité d'obtenir $Y^{\tau(\pi)}$ dépend de la probabilité abstraite $p^{\tau(\pi)}$.

Il se peut également qu'une variable d'état dépende des régions d'arrivée de la politique π. Considérons par exemple une variable `autonomie` qui représente l'énergie restante du robot. La consommation d'énergie dépend de la politique suivie, et donc des régions de départ $\tau(\pi)$ et d'arrivée $\zeta(\pi)$. Ainsi, une feuille de l'arbre de probabilité de la variable `autonomie` est une fonction de $\tau(\pi)$ et $\zeta(\pi)$, et elle a entre autres pour parent le nœud $\tilde{X}'_p$. Le nœud $\tilde{X}_p$ n'est pas nécessairement un nœud parent car π est implicitement définie pour la seule valeur $\tau(\pi)$ de $\tilde{X}_p$.

Par conséquent, nous supposons que les probabilités d'obtenir les différentes valeurs des variables d'état, stockées dans les feuilles des arbres de probabilité, sont des fonctions algébriques et formelles de :

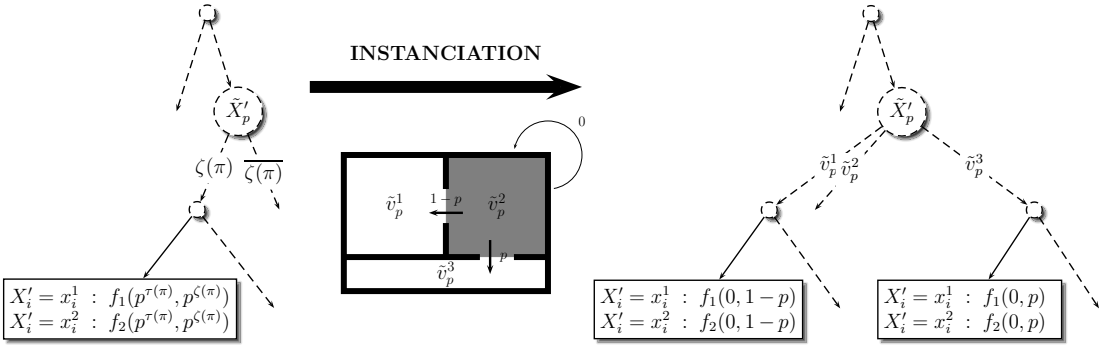


FIG. 5.5 – Exemple de feuille générique de l'arbre de probabilité d'une variable binaire, ainsi qu'une de ses instanciations possibles pour la région $\tilde{v}_p^2$ d'un sous-espace de navigation constitué de 3 régions

- $p^{\tau(\pi)}$: probabilité d'obtenir la valeur de la partition où π est applicable
- $p^{\zeta(\pi)}$: si $\tilde{X}'_p$ est un nœud parent, probabilité d'obtenir la valeur de la partition correspondant au sous-arbre de $\tilde{X}'_p$ dans lequel se trouve la feuille évaluée

Les feuilles sont évaluées lors de l'instanciation d'une politique π, à l'aide d'une bibliothèque logicielle de calcul formel[78, 79]. La figure 5.5 représente un exemple de feuille générique de l'arbre de probabilité d'une variable binaire, ainsi qu'une de ses instanciations possibles.

5.1.5 Paramétrage des récompenses

Comme pour les probabilités de transiter entre les différentes régions du partitionnement $\tilde{V}_p$, chaque politique locale est associée à une fonction $R^\pi \ : \ \tau(\pi) \times \zeta(\pi) \rightarrow \mathbb{R}$, qui indique la récompense moyenne accumulée en suivant la politique π. Cette récompense moyenne peut être calculée à partir des probabilités de transition P^π entre les différentes partitions [23, 24, 57].

Nous supposons que les feuilles de l'arbre de récompense générique sont des fonctions algébriques formelles de :

- $r^{\tau(\pi)}$: récompense moyenne accumulée en restant dans la région $\tau(\pi)$
- $r^{\zeta(\pi)}$: si $\tilde{X}'_p$ est un nœud parent, récompense moyenne accumulée en obtenant la valeur de la partition correspondant au sous-arbre de $\tilde{X}'_p$ dans lequel se trouve la feuille évaluée

La figure 5.6 représente un exemple de feuille générique de l'arbre de récompense, ainsi qu'une de ses instanciations possibles.

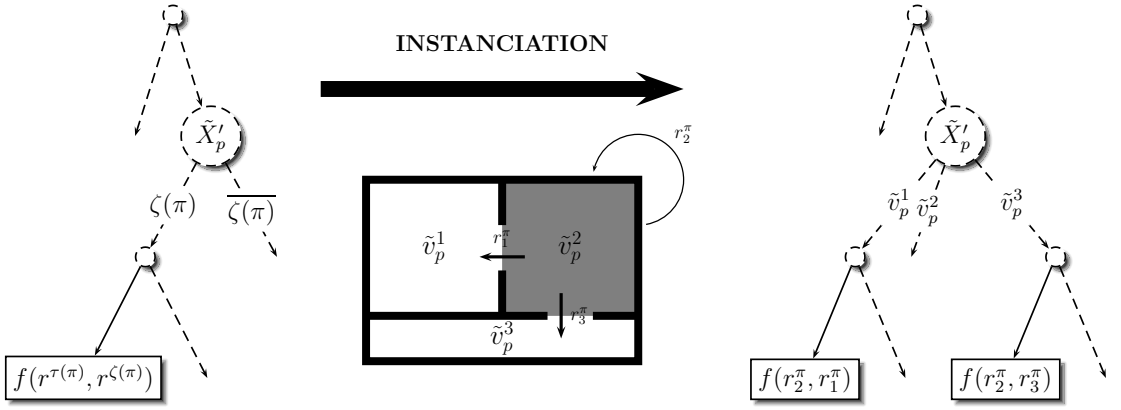


Fig. 5.6 – Exemple de feuille générique de l'arbre de récompense, ainsi qu'une de ses instanciations possibles pour la région $\tilde{v}_p^2$ d'un sous-espace de navigation constitué de 3 régions

5.1.6 Paramétrisation par classes de politiques locales

Pour certains types de problèmes, il est insuffisant de modéliser les transitions avec une seule politique générique. Considérons par exemple le problème d'exploration décrit dans la figure 5.3. Les variables $\left(Y^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{V}_p}$ représentent des objectifs à atteindre dans chaque région du sous-espace de navigation. Toutes les politiques qui consistent à atteindre un objectif $Y^{\tilde{v}_p}$ particulier ne sont applicables que sur le sous-espace $Y^{\tilde{v}_p} = \texttt{false}$. Une fois que cet objectif est atteint, les politiques qui consistent à sortir de la région ont été optimisées avec un objectif absent de la région, si bien qu'elles sont alors définies sur le sous-espace complémentaire $Y^{\tilde{v}_p} = \texttt{true}$. Ainsi, pour ce problème, l'arbre de masque est paramétré par des classes de politiques locales dont la relation d'équivalence est : «deux politiques sont équivalentes si elles s'appliquent sur le même sous-espace d'états».

Notons $\mathcal{C}_p$ l'ensemble des classes de politiques locales par une relation d'équivalence $\mathcal{R}_p$ connue lors de la modélisation du problème. Au lieu de définir un seul DBN générique, nous définissons autant de DBN génériques que de classes de politiques locales. Chaque DBN est paramétré par les politiques locales de la classe d'équivalence qui lui est associé. L'intérêt de notre modèle générique de DBN n'est généralement pas perdu car le nombre de classes de politiques locales devrait être faible par rapport au nombre de politiques locales considérées, pour la plupart des problèmes. La figure 5.7 montre un exemple de deux arbres de masque définis chacun pour des classes différentes de politiques locales.

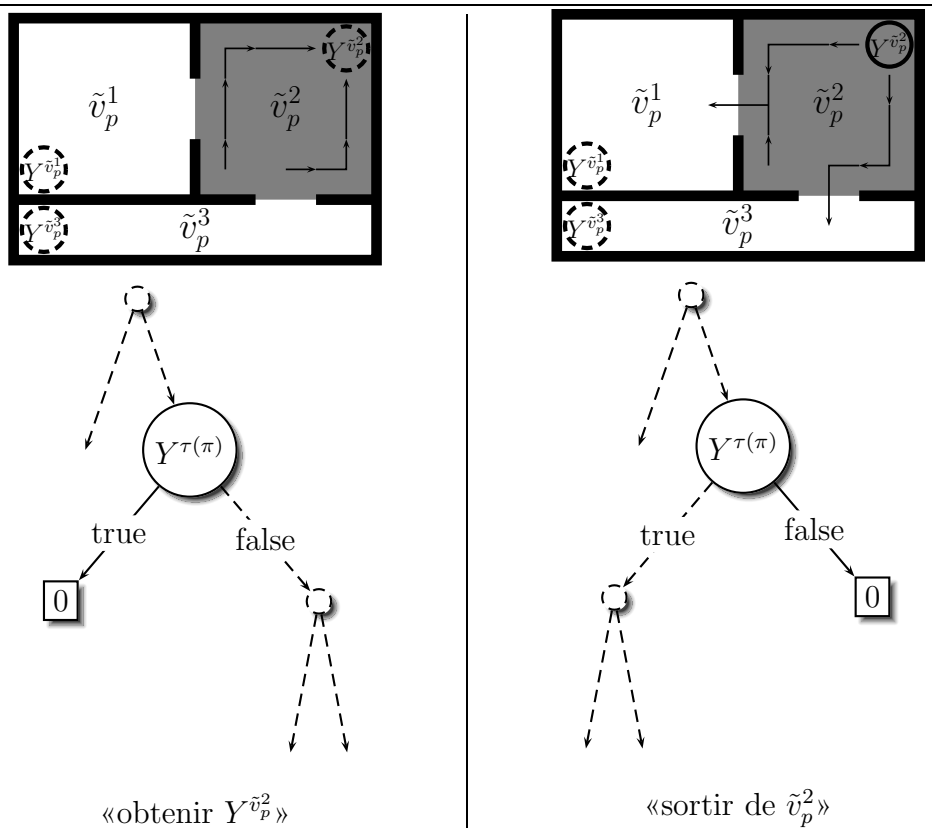


FIG. 5.7 – Exemple d'arbres de masque définis pour des classes différentes de politiques locales

5.2 Algorithme d'instanciation automatique des réseaux bayésiens dynamiques

5.2.1 Algorithme principal

L'algorithme principal 16 (fonction `InstancieMDP`) prend en entrée un MDP générique dont les actions sont des classes de politiques locales, et les arbres de décision sont définis chacun pour une classe de politiques locales donnée. Il itère sur toutes les politiques du pool Π_p de politiques locales afin d'instancier tous les arbres de décision du MDP générique.

Dans la section précédente, nous avons introduit deux variables symboliques $Y^{"\tau(\pi)"}$ et $Y^{"\overline{\tau(\pi)}"}$ qui représentent une variable définie pour chaque valeur de la variable réduite (une fois instanciée). Nous généralisons ce formalisme à un ensemble de telles paires de variables symboliques. Nous notons $\mathcal{Y}_p$ l'ensemble des noms de variables symboliques : dans le cas d'une seule paire $\left(Y^{"\tau(\pi)"}, Y^{"\overline{\tau(\pi)}"}\right)$ de variables

Algorithme 16 : Fonction `InstancieMDP`

Données : $\langle \tilde{\mathcal{V}}, \mathcal{C}_p, (T^\pi_X)_{\substack{\pi \in \mathcal{C}_p \\ X \in \mathcal{V}}}, (R^\pi)_{\pi \in \mathcal{C}_p} \rangle$ (MDP générique), τ, ζ, Π_p, $\mathcal{Y}_p$

Résultat : $\left\langle \overline{\mathcal{V}}, \Pi_p, \left(\overline{T}^\pi_X\right)_{\substack{\pi \in \Pi_p \\ X \in \overline{\mathcal{V}}}}, \left(\overline{R}^\pi\right)_{\pi \in \Pi_p} \right\rangle$ (MDP instancié)

début

  pour $X \notin \mathcal{Y}_p$ **faire**
    $\overline{\mathcal{V}} \leftarrow \overline{\mathcal{V}} \cup V_X$;
  pour $Y \in \mathcal{Y}_p$ **faire**
    pour $\tilde{v}_p \in \tilde{V}_p$ **faire**
      $\overline{\mathcal{V}} \leftarrow \overline{\mathcal{V}} \cup Y^{\tilde{v}_p}$;
  pour $\pi \in \Pi_p$ **faire**
    pour $X \notin \mathcal{Y}_p$ **faire**
      $\overline{T}^\pi_X \leftarrow$ `InstancieArbre`$(T^\pi_X, \pi, \tau, \zeta)$;
    pour $Y \in \mathcal{Y}_p$ **faire**
      $\left(\overline{T}^\pi_{Y^{\tilde{v}_p}}\right)_{\tilde{v}_p \in \tilde{V}_p} \leftarrow$ `InstancieArbresYp`$(T^\pi_{Y^{"\tau(\pi)"}}, T^\pi_{Y^{"\overline{\tau(\pi)}"}}, \pi, \tau, \zeta)$;
    $\overline{R}^\pi \leftarrow$ `InstancieArbre`$(R^\pi, \pi, \tau, \zeta)$;
  retourner $\left\langle \overline{\mathcal{V}}, \Pi_p, \left(\overline{T}^\pi_X\right)_{\substack{\pi \in \Pi_p \\ X \in \overline{\mathcal{V}}}}, \left(\overline{R}^\pi\right)_{\pi \in \Pi_p} \right\rangle$;

fin

symboliques, $\mathcal{Y}_p = \{Y\}$.

La fonction `InstancieMDP` sépare l'ensemble des variables d'état de celles de $\mathcal{Y}_p$. Pour chaque politique locale de Π_p, elle appelle les fonctions :

- `InstancieArbre` : instancie l'arbre de récompense, ou l'arbre de probabilité d'une variable d'état autre que $\left(Y^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{V}_p}$; par abus de notation, T^π_X est l'arbre de probabilité générique de la variable d'état X *pour la classe de politiques locales à laquelle appartient* π
- `InstancieArbresYp` : instancie les arbres de probabilité des variables $\left(Y^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{V}_p}$ à partir des arbres de probabilité des deux variables abstraites $Y^{"\tau(\pi)"}$ et $Y^{"\overline{\tau(\pi)}"}$

5.2.2 Fonction `InstancieArbresYp`

La fonction `InstancieArbresYp` (cf. algorithme 17) instancie les arbres de probabilité d'une paire de variables symboliques $\left(Y^{"\tau(\pi)"}, Y^{"\overline{\tau(\pi)}"}\right)$, où $Y \in \mathcal{Y}_p$. Les variables $\left(T^\pi_{Y^{\tilde{v}_p}}\right)_{\tilde{v}_p \in \tilde{V}_p}$ sont définies pour chaque valeur de $\tilde{v}_p$, si bien qu'autant d'arbres

de probabilité que de valeurs abstraites de la variable réduite sont générés.

La paire $\left(Y^{"\tau(\pi)"}, Y^{"\overline{\tau(\pi)}"}\right)$ n'est pas évaluée : les guillemets signifient que les expressions "$\tau(\pi)$" et "$\overline{\tau(\pi)}$" ne sont pas des fonctions, mais des symboles ASCII reconnaissables par le parser. Ils peuvent également être des identifiants des membres de la structure Y (ou variable objet Y dans un langage orienté objet).

Algorithme 17 : Fonction `InstancieArbresYp`

Données : $T_{Y^{"\tau(\pi)"}}$, $T_{Y^{"\overline{\tau(\pi)}"}}$, π, τ, ζ
Résultat : $\left(T^{\pi}_{Y^{\tilde{v}_p}}\right)_{\tilde{v}_p \in \tilde{V}_p}$
début
  $T^{\pi}_{Y^{\tau(\pi)}} \leftarrow$ `InstancieArbre`$(T_{Y^{"\tau(\pi)"}}, \pi, \tau, \zeta)$;
  pour $\tilde{v}_p \in \overline{\tau(\pi)}$ **faire**
    $T^{\pi}_{Y^{\tilde{v}_p}} \leftarrow$ `InstancieArbre`$(T_{Y^{"\overline{\tau(\pi)}"}}, \pi, \tau, \zeta)$;
  retourner $\left(T^{\pi}_{Y^{\tilde{v}_p}}\right)_{\tilde{v}_p \in \tilde{V}_p}$;
fin

5.2.3 Fonction `InstancieArbre`

La fonction `InstancieArbre` (cf. algorithme 18) parse un arbre en analysant chaque élément et en créant les élements appropriés de l'arbre instancié correspondant. L'algorithme implémenté est récursif car les éléments de l'arbre initial et de l'arbre instancié ne sont pas nécessairement identiques (à cause des sous-arbres symboliques des nœuds $\tilde{X}_p$ et $\tilde{X}'_p$. Un algorithme itératif imposerait de garder en mémoire la liste des sous-arbres lus dans l'arbre générique et celle des sous-arbres créés dans l'arbre instancié. L'itération sur ces deux listes est complexe car leurs éléments respectifs ne correspondent pas nécessairement.

Le paramètre $\tilde{v}'_p$ est optionnel, et vaut -1 par défaut. Il indique la valeur d'un éventuel nœud $\tilde{X}'_p$ parent. Si $\tilde{X}'_p$ est un nœud parent, $\tilde{v}'_p$ est la valeur correspondant au sous-arbre de $\tilde{X}'_p$ auquel T appartient. Une valeur de -1 signifie que le nœud $\tilde{X}'_p$ n'est pas parent. Cette valeur est transmise à toutes les fonctions qui appellent récursivement `InstancieArbre`.

La fonction `InstancieArbre` appelle les fonctions suivantes :

- `InstancieFeuille` : instancie une feuille en testant si l'arbre $\mathcal{T}$ est un arbre de probabilité ou de récompense,
- `InstancieNoeud` : instancie un nœud en distinguant les nœuds symboliques $\left(Y^{"\tau(\pi)"}\right)_{Y \in \mathcal{Y}_p}$ des autres nœuds,

Algorithme 18 : Fonction `InstancieArbre` (récursive)

Données : T (arbre de décision générique), π, τ, ζ, $[\tilde{v}'_p = -1]$
Résultat : T^π (arbre de décision instancié)
début
  si $T.racine().type() = feuille$ **alors**
    $T^\pi \leftarrow$ `InstancieFeuille`$(T.racine(), \pi, \tau, \zeta, \tilde{v}'_p)$;
  sinon
    $T^\pi \leftarrow$ `InstancieNoeud`$(T.racine(), \pi, \tau)$;
    suivant $T.racine()$ **faire**
      cas où $\tilde{X}_p$
        $T^\pi \leftarrow$ `InstancieSousArbresXp`$(T.racine(), T^\pi, \pi, \tau, \zeta, \tilde{v}'_p)$;
      cas où $\tilde{X}'_p$
        $T^\pi \leftarrow$ `InstancieSousArbresXpp`$(T.racine(), T^\pi, \pi, \tau, \zeta)$;
      autres cas
        pour $sous\_arbre \in T.fils()$ **faire**
          $T^\pi.fils().enfiler($`InstancieArbre`$(sous\_arbre, \pi, \tau, \zeta, \tilde{v}'_p))$;
  retourner T^π;
fin

- `InstancieSousArbresXp` : instancie les sous-arbres d'un nœud $\tilde{X}_p$ en appelant de nouveau `InstancieArbre` sur chaque sous-arbre,
- `InstancieSousArbresXpp` : instancie les sous-arbres d'un nœud $\tilde{X'}_p$ en appelant de nouveau `InstancieArbre` sur chaque sous-arbre ; la valeur $\tilde{v}'_p$ n'est pas transmise car elle vaut nécessairement -1 (sinon, le nœud $\tilde{X}'_p$ aurait pour parent lui-même, ce qui est impossible).

Fonction `InstancieFeuille`

La fonction `InstancieFeuille` (cf. algorithme 19) teste si l'arbre auquel appartient la feuille générique est un arbre de probabilité ou de récompense. Elle appelle la fonction `EvalueExpression` d'une bibliothèque logicielle quelconque de calcul formel [80, 78, 79] afin d'évaluer la feuille générique en fonction des variables "$[p, r]^{\tau(\pi)}$" et "$[p, r]^{\zeta(\pi)}$". Ces variables sont affectées des valeurs (probabilité ou récompense) des macro-transitions entre les différentes partitions de $\tilde{V}_p$ pour la politique π. La deuxième variable est optionnelle : elle dépend de la présence d'un nœud parent $\tilde{X}'_p$.

Algorithme 19 : Fonction `InstancieFeuille`

Données : f (feuille générique), π, τ, ζ, $[\tilde{v}'_p = -1]$
Résultat : f^π (feuille instanciée)
début
  si $f.arbre().type() = probabilite$ **alors**
    $p^{\tau(\pi)} \leftarrow T^\pi(\tau(\pi), \tau(\pi))$;
    si $\tilde{v}'_p \neq -1$ **alors**
      $p^{\zeta(\pi)} \leftarrow T^\pi(\tau(\pi), \tilde{v}'_p)$;
    $f^\pi \leftarrow$ `EvalueExpression`$(f, "p^{\tau(\pi)}" = p^{\tau(\pi)}, ["p^{\zeta(\pi)}" = p^{\zeta(\pi)}])$;
  sinon
    $r^{\tau(\pi)} \leftarrow R^\pi(\tau(\pi), \tau(\pi))$;
    si $\tilde{v}'_p \neq -1$ **alors**
      $r^{\zeta(\pi)} \leftarrow R^\pi(\tau(\pi), \tilde{v}'_p)$;
    $f^\pi \leftarrow$ `EvalueExpression`$(f, "r^{\tau(\pi)}" = r^{\tau(\pi)}, ["r^{\zeta(\pi)}" = r^{\zeta(\pi)}])$;
  retourner f^π;
fin

Fonction `InstancieNoeud`

La fonction `InstancieNoeud` (cf. algorithme 20) teste si le nœud générique n contient une variable symbolique de $\mathcal{Y}_p$. Si tel est le cas, le nœud instancié n^π est affecté de la variable de $\mathcal{Y}_p$ définie dans la région $\tau(\pi)$. Par abus de notation, n représente ici le nom de la variable symbolique contenue dans le nœud n. Par exemple, pour les deux variables symboliques $Y^{\tau(\pi)}$ et $Y^{\overline{\tau(\pi)}}$: $n = "Y"$. Il n'y a pas de confusion entre ces deux variables symboliques car la variable $Y^{\overline{\tau(\pi)}}$ n'est pas autorisée dans un arbre de décision générique.

Algorithme 20 : Fonction `InstancieNoeud`

Données : n (nœud générique), π, τ
Résultat : n^π (nœud instancié)
début
  si $n \in \mathcal{Y}_p$ **alors**
    $n^\pi \leftarrow n^{\tau(\pi)}$;
  sinon
    $n^\pi \leftarrow n$;
  retourner n^π;
fin

Fonction `InstancieSousArbresXp`

La fonction `InstancieSousArbresXp` (cf. algorithme 21) prend en entrée la racine du sous-arbre générique T en cours d'analyse, et celle du sous-arbre T^π en cours d'instanciation. Ainsi, afin de faciliter la compréhension de l'algorithme, T et T^π sont en fait des nœuds. Par abus de notation, la valeur retournée T^π est l'arbre instancié dont la racine est notée initialement T^π. L'arbre T^π est instancié en appelant la fonction `InstancieArbre` (cf. algorithme 18) sur le fils correspondant à la région $\tau(\pi)$ (les autres sous-arbres sont nuls car π n'est définie que dans $\tau(\pi)$).

Algorithme 21 : Fonction `InstancieSousArbresXp`

Données : T (nœud générique), T^π (nœud instancié), π, τ, ζ, $[\tilde{v}'_p = -1]$
Résultat : T^π (arbre instancié)
début
  $sous\_arbre \leftarrow T.fils(\text{"}\tau(\pi)\text{"})$;
  pour $\tilde{v}_p \in \tilde{V}_p$ **faire**
    si $\tilde{v}_p = \tau(\pi)$ **alors**
      $T^\pi.fils().enfiler(\texttt{InstancieArbre}(sous\_arbre, \pi, \tau, \zeta, \tilde{v}'_p))$;
    sinon
      $T^\pi.fils().enfiler(feuille\_nulle)$;
  retourner T^π;
fin

Fonction `InstancieSousArbresXpp`

La fonction `InstancieSousArbresXpp` (cf. algorithme 22) appelle la fonction récursive `InstancieArbre` (cf. algorithme 18) sur chacun des sous-arbres du nœud T^π, afin d'instancier l'arbre T^π retourné (abus de notation). L'argument optionnel $(\tilde{v}'_p)$ de `InstancieArbre` est initialisé lors de l'instanciation de chaque sous-arbre, puisque $T^\pi = \tilde{X}'_p$ dans cette fonction.

Algorithme 22 : Fonction `InstancieSousArbresXpp`

Données : T (nœud générique), T^{π} (nœud instancié), π, τ, ζ
Résultat : T^{π} (arbre instancié)
début
  $sa_1 \leftarrow T.fils(\text{"}\underline{\zeta(\pi)}\text{"})$;
  $sa_2 \leftarrow T.fils(\text{"}\overline{\zeta(\pi)}\text{"})$;
  pour $\tilde{v}'_p \in \tilde{V}'_p$ **faire**
    si $\tilde{v}'_p \in \zeta(\pi)$ **alors**
      $T^{\pi}.fils().enfiler(\texttt{InstancieArbre}(sa_1, \pi, \tau, \zeta, \tilde{v}'_p))$;
    sinon
      $T^{\pi}.fils().enfiler(\texttt{InstancieArbre}(sa_2, \pi, \tau, \zeta, \tilde{v}'_p))$;
  retourner T^{π};
fin

Chapitre 6

Modélisation de missions ReSSAC avec des Réseaux Bayésiens Dynamiques Génériques et Hiérarchiques

6.1 Outil de modélisation

Nous avons conçu un outil informatique programmé avec la bibliothèque graphique Bakery [81] permettant de modéliser et de résoudre une mission ReSSAC [4, 5] à l'aide de notre modèle générique de MDP factorisé. La partie modélisation comprend un éditeur du sous-espace de navigation et un éditeur du DBN générique.

La bibliothèque Bakery est une architecture C++ orientée objet Document/Vue [82], c'est-à-dire que le traitement des données (le document) est séparé du rendu à l'écran (la vue). Dans Bakery, le document est encodé au format XML [83] et la vue est programmée en Gtkmm [84]. La syntaxe d'une mission ReSSAC (DTD, voir [83]) et l'architecture de notre application sont décrits dans l'annexe A (page 271).

6.1.1 Modélisation du sous-espace de navigation

L'éditeur du sous-espace de navigation (cf. figure 6.1) est une vue 3D programmée en OpenGL [85] et GtkGLExtmm [86]. Elle permet de définir les variables `point de passage` (variable microscopique X_p) et `région` (variable macroscopique $\tilde{X}_p$), ainsi que les transitions entre les points de passage. Cette vue permet donc de définir le sous-MDP de navigation et son partitionnement en régions.

Les points de passage de l'hélicoptère ainsi que les transitions stochastiques entre ces points de passage sont définis dans une boîte de dialogue (cf. figure 6.2). L'uti-

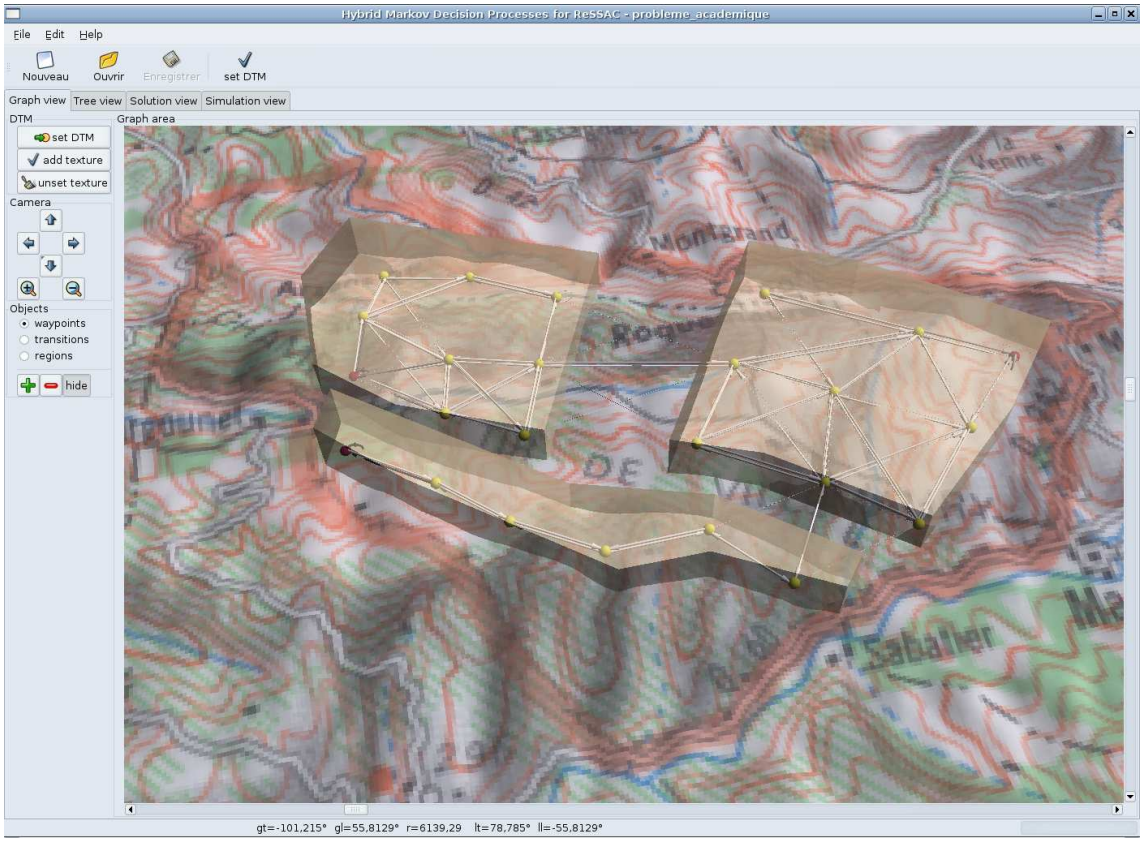


FIG. 6.1 – Capture d'écran de l'éditeur du sous-espace de navigation

lisateur peut définir les coordonnées 3D du point de passage ainsi que la liste, pour chaque action, des transitions partant du point de passage édité. Un point de passage peut être un sous-but à atteindre (sphères foncées dans la figure 6.1). Un sous-but doit être défini dans chaque région $\tilde{v}_p$: il correspond à la variable $Y^{\tilde{v}_p}$ définie pour la valeur $\tilde{v}_p$ de la variable `région` $\tilde{X}_p$.

Les régions sont définies à l'aide d'une boîte de dialogue spécifique (cf. figure 6.3). Chaque région est délimitée par des points de contrôle (petits cubes) éditables. Un algorithme d'enroulement calcule automatiquement l'ensemble des points de passage contenus à l'intérieur de la région : un point de passage W appartient à une région r si :

$$\left| \sum_{i=1}^{|\mathcal{C}_r|} \left(\overrightarrow{WC_i}, \overrightarrow{WC_{i+1}} \right) \right| = 2\pi$$

où $\mathcal{C}_r = (C_i)_{1 \leqslant i \leqslant |\mathcal{C}_r|}$ est l'ensemble des points de contrôle de la région r, et avec la convention $C_{|\mathcal{C}_r|+1} = C_1$.

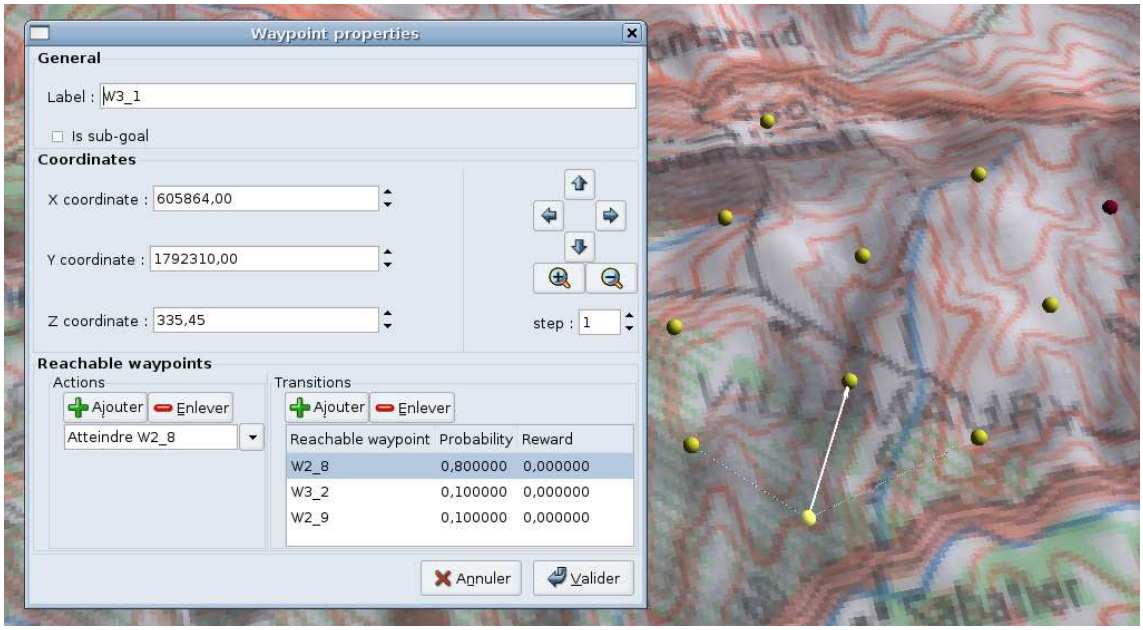


FIG. 6.2 – Capture d'écran de l'éditeur de points de passage

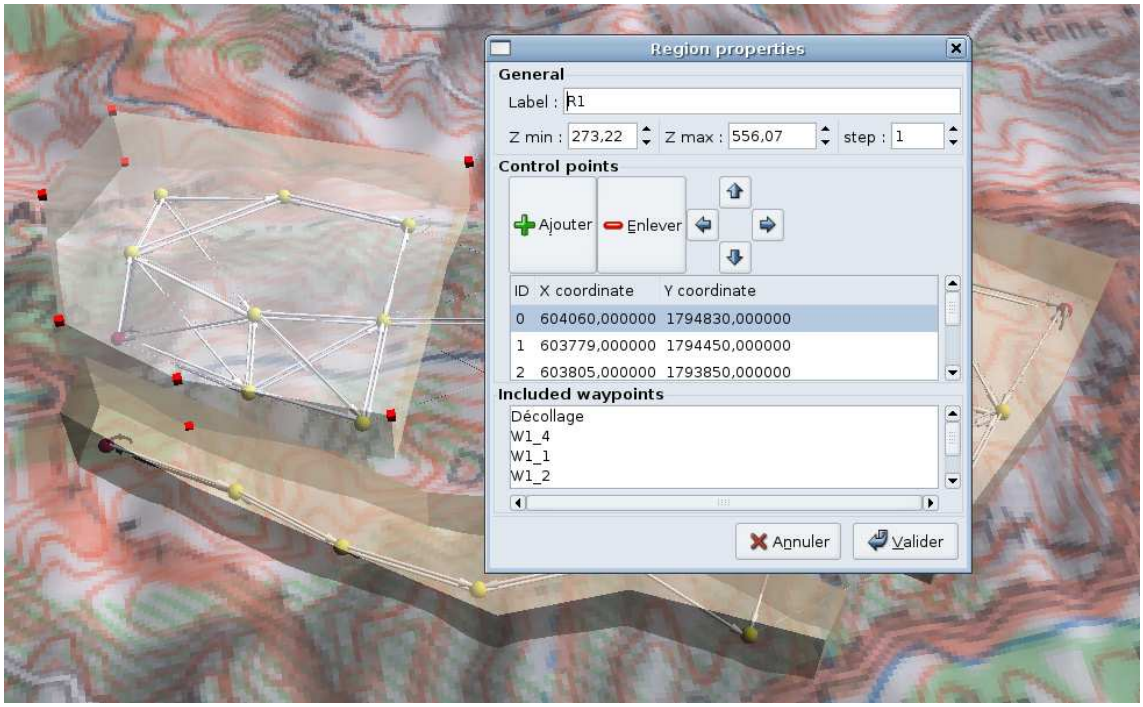


FIG. 6.3 – Capture d'écran de l'éditeur de régions

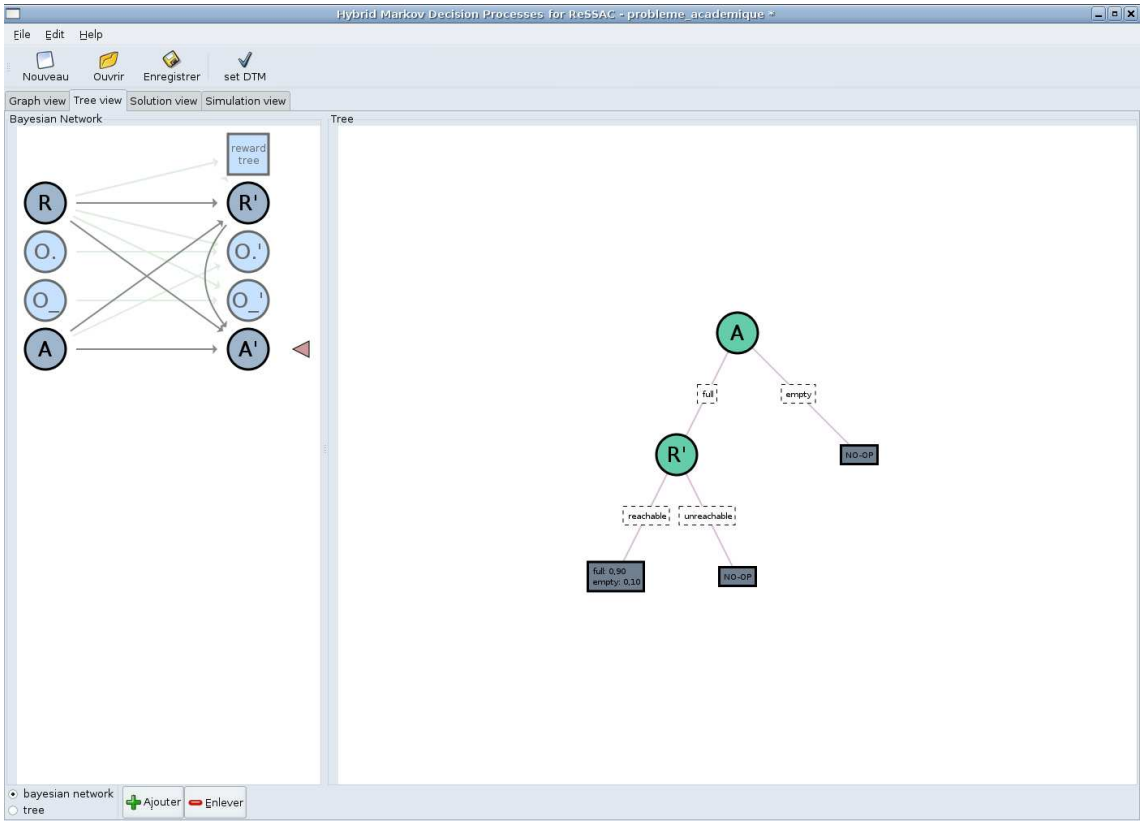


FIG. 6.4 – Capture d'écran de l'éditeur du DBN générique

6.1.2 Modélisation du DBN générique

Le DBN générique (cf. figure 6.4) est séparé en deux sous-vues :

- représentation graphique du DBN générique (dépendances entre les variables d'état),
- représentation graphique de l'arbre (récompense ou probabilité) générique sélectionné dans la sous-vue du DBN générique.

La vue du DBN générique indique que tous les arbres de décision dépendent de la variable R alors qu'aucun nœud R n'est présent dans les arbres de décision. En effet, la politique paramètre π n'est définie que dans la région $\tau(\pi)$ où elle s'applique. Le logiciel gère en fait un arbre de masque qui indique que π n'est applicable que dans la région $R = \tau(\pi)$, ce qui dispense l'utilisateur d'insérer inutilement des nœuds R dans les arbres de décision. La dépendance en R indiquée dans le DBN générique rappelle que l'arbre de masque est automatiquement défini. Cette approche ne permet cependant pas de définir des arbres de masque spécifiques.

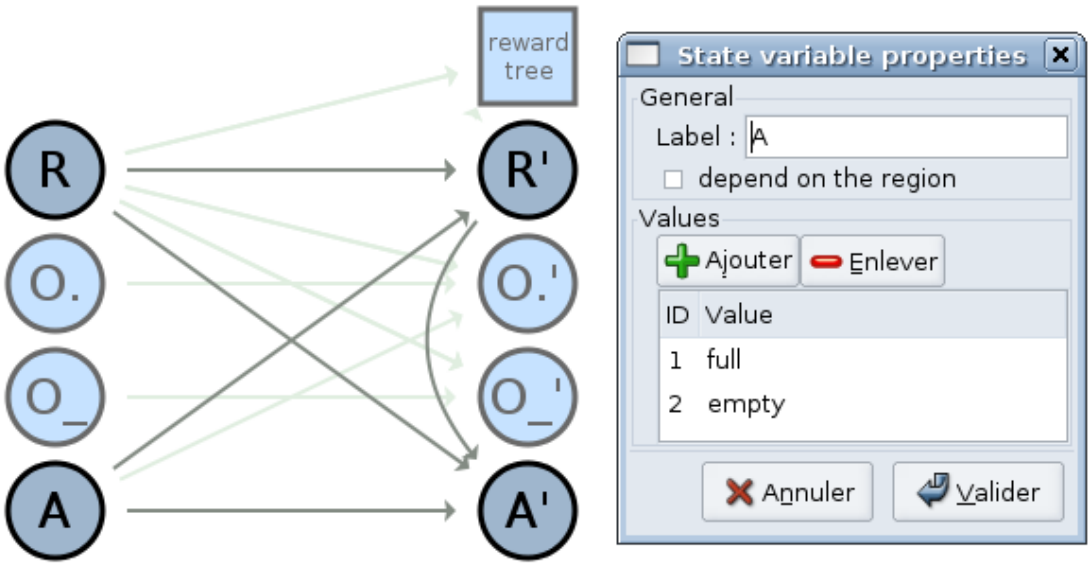


FIG. 6.5 – Capture d'écran de l'éditeur de variable d'état

L'utilisateur peut rajouter à volonté des variables d'état d'arité quelconque (cf. figure 6.5), et spécifier si la nouvelle variable d'état dépend de la région paramètre du DBN générique, c'est-à-dire si elle appartient à $\mathcal{Y}_p$. Pour chaque nouvelle variable d'état Y de $\mathcal{Y}_p$, le logiciel crée deux variables symboliques $Y.$ ($Y^{\tau(\pi)}$) et $Y\_$ ($Y^{\overline{\tau(\pi)}}$). Par défaut, une mission contient les variables symboliques $O.$ et $O\_$ qui représentent les objectifs à atteindre dans chaque région (points de passage sous-buts).

Les éléments des arbres de décision sont modélisés à l'aide d'une boîte de dialogue qui permet de définir à la fois des nœuds ou des feuilles (cf. figure 6.6). Les feuilles sont des expressions formelles de 6 variables qui dépendent de la politique paramètre du DBN générique :

- $Lp.$, probabilité de rester dans la région paramètre ($p^{\tau(\pi)}$)
- $Lv.$, récompense moyenne accumulée en restant dans la région paramètre ($r^{\tau(\pi)}$)
- $Lp\_$, probabilité de sortir dans la région paramètre ($1 - p^{\tau(\pi)}$)
- $Lv\_$, récompense moyenne accumulée en sortant de la région paramètre ($r^{\overline{\tau(\pi)}}$)
- Lp : , si R' est nœud parent (*ou* si l'arbre édité est l'arbre de probabilité de R), probabilité d'aller dans la région du nœud R' parent ($p^{\zeta(\pi)}$)
- Lv : , si R' est nœud parent (*ou* si l'arbre édité est l'arbre de probabilité de R), récompense moyenne accumulée en allant dans la région du nœud R' parent ($r^{\zeta(\pi)}$)

Une feuille `NoOp` d'un arbre de probabilité d'une variable X_i indique que les valeurs de la variable X_i restent inchangées.

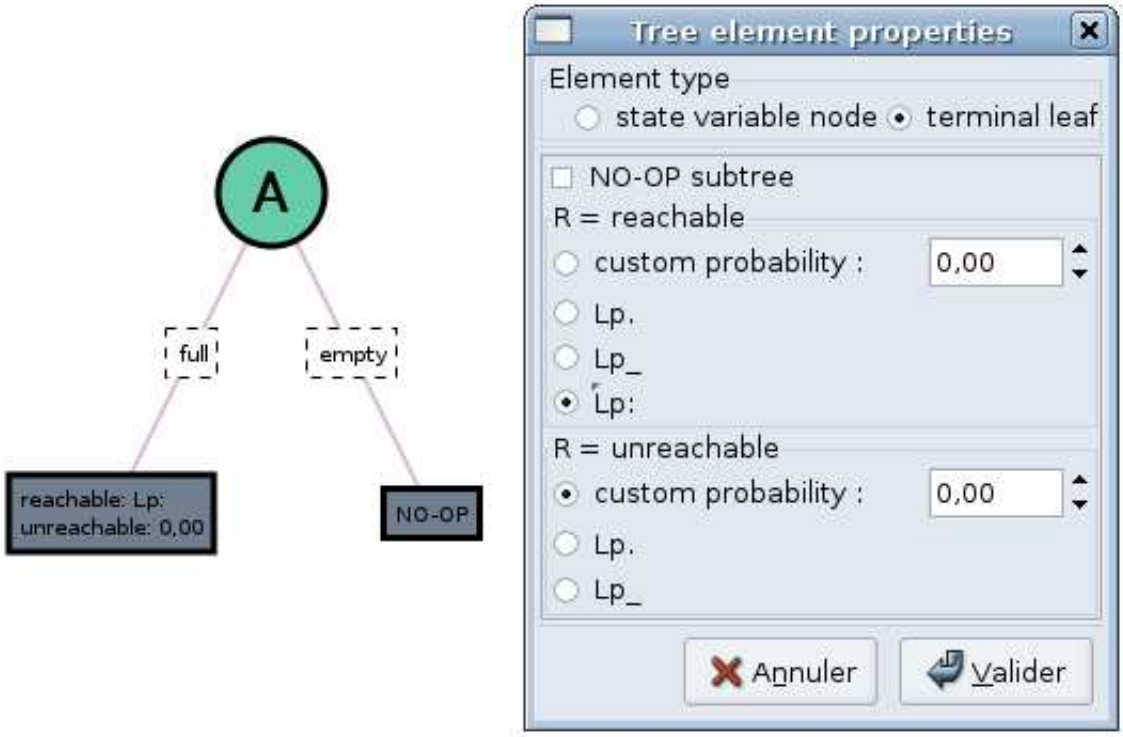


FIG. 6.6 – Capture d'écran de l'éditeur d'élément des arbres de décision

6.1.3 Modélisation des arbres de décision génériques

Dans ce paragraphe, nous présentons un exemple d'arbres de décision définis à l'aide de notre interface graphique, afin de montrer la faisabilité de notre modèle générique. La mission dépend bien entendu des arbres de décision définis.

Pour chaque région, nous considérons deux classes de politiques locales : une visant à atteindre l'objectif interne de la région (variable $O. = O^{\tau(\pi)}$), l'autre consistant à sortir de la région une fois cet objectif atteint. Le seul arbre paramétré par les classes de politiques locales est l'arbre de masque, caché à l'utilisateur (cf. figure 6.7).

Les récompenses de la mission sont définies au niveau du sous-espace de navigation. Elles sont positives lorsque l'hélicoptère survole un point de passage sous-but, pour la classe de politiques visant à atteindre le sous-but interne à la région paramètre. Pour la classe de politiques consistant à sortir de la région, la récompense accumulée en suivant la politique paramètre peut être négative si certaines zones survolées présentent un danger pour le drone. Ainsi, au niveau abstrait factorisé, la récompense dépend de la région d'arrivée R' de la politique paramètre (cf. figure 6.8). La récompense accumulée $Lv := r^{\zeta(\pi)}$ en survolant la région $\tau(\pi)$ est fournie

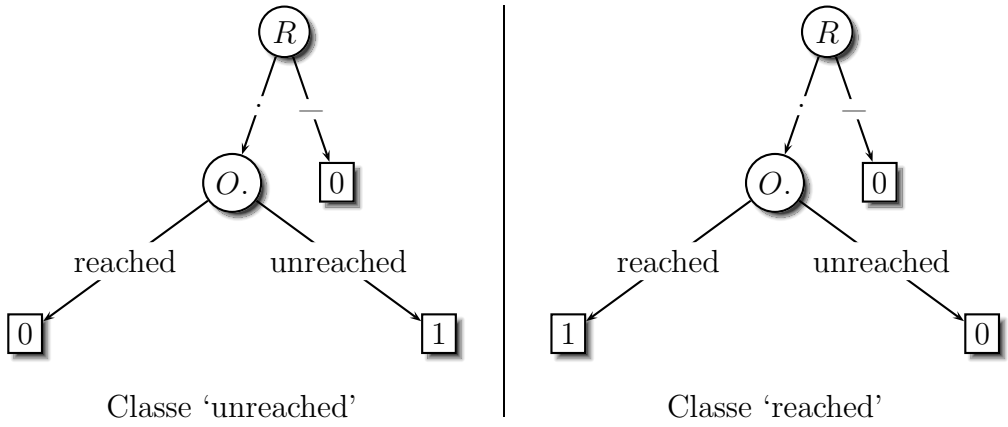


FIG. 6.7 – Arbre de masque paramétré par les classes de politiques locales $\mathcal{C}_X = \{\text{reached}, \text{unreached}\}$, caché à l'utilisateur

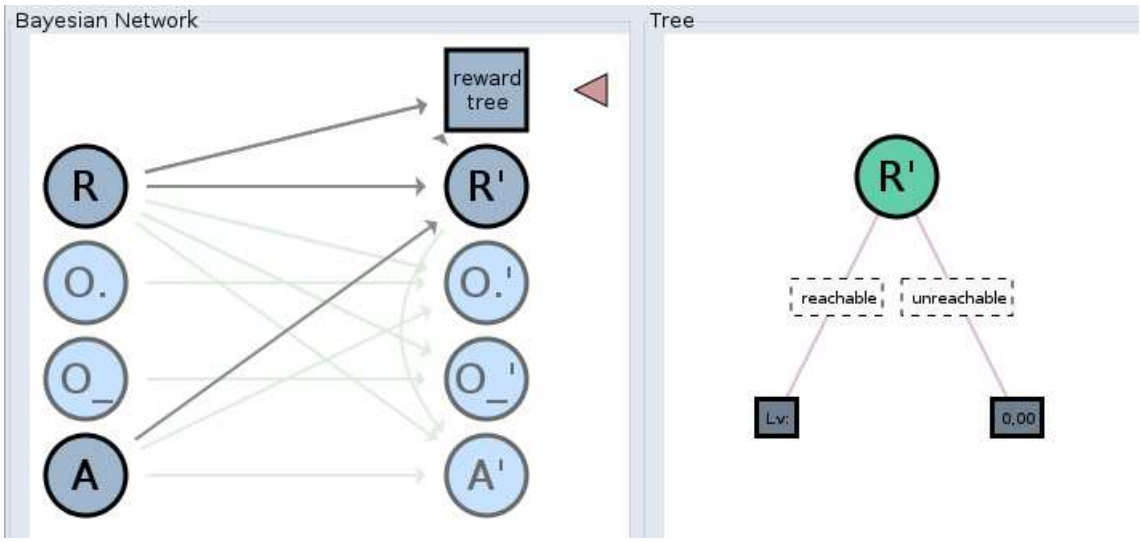


FIG. 6.8 – Capture d'écran de l'éditeur de l'arbre de récompense générique

avec chaque politique locale.

L'arbre probabilité de la variable R (cf. figure 6.9) dépend de la variable d'autonomie A. Pour simplifier, nous supposons qu'une région est atteignable si l'autonomie restante du drone est suffisante. Si tel est le cas, la probabilité d'atteindre chaque région est la probabilité symbolique $Lp := p^{\zeta(\pi)}$ qui est donnée avec chaque politique locale. Si l'autonomie restante n'est pas suffisante, le drone reste dans la région où il est (feuille `NoOp`).

L'objectif interne d'une région ne peut être atteint que s'il ne l'a pas encore été (cf. figure 6.10). D'après l'arbre de masque (cf. figure 6.7), le sous-arbre 'unreached'

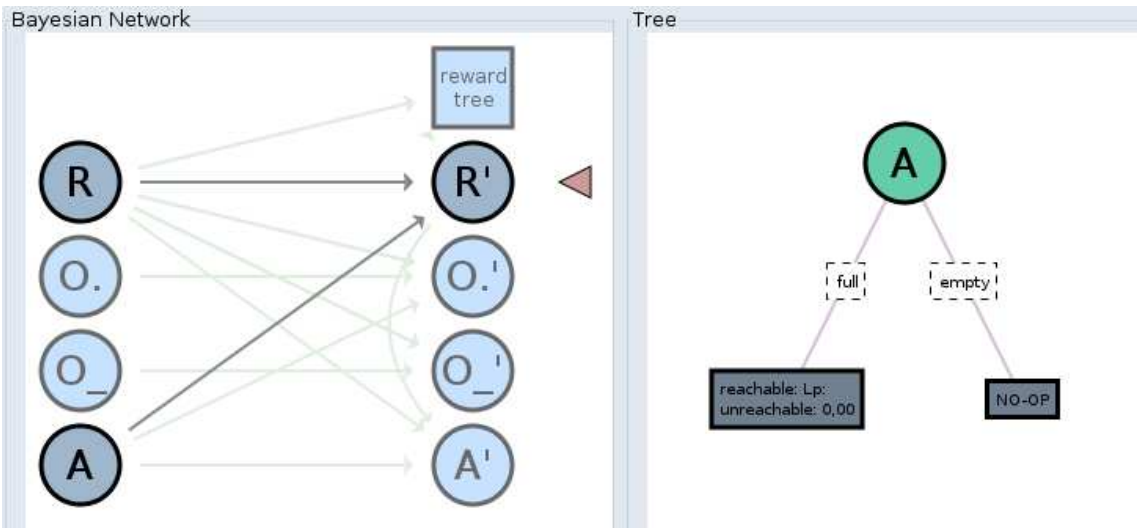


FIG. 6.9 – Capture d'écran de l'éditeur de l'arbre de probabilité générique de la variable $\tilde{X}_p = R$

du nœud $O.$ correspond à une politique visant à atteindre l'objectif $O.$. Ainsi, nous supposons que l'objectif interne à la région paramètre est atteint lorsque le drone reste dans cette région : si l'autonomie est suffisante, l'objectif interne est atteint avec une probabilité $Lp. = p^{\tau(\pi)}$.

Les objectifs externes à la région paramètre ne sont pas atteignables. Dans le cas où un objectif se trouverait à une sortie de cette région **et** que la politique paramètre (une fois instanciée) consiste à rejoindre cette sortie, il pourrait certes être atteint par cette politique, mais il est concevable également qu'une politique à un coup de la région atteinte permet d'atteindre immédiatement cet objectif s'il ne l'a pas été par la politique précédente. Ainsi, il n'est pas restrictif de considérer qu'aucun objectif externe ne peut être par la politique paramètre, ce qui signifie que l'arbre de probabilité de la variable symbolique $O\_ = O^{\overline{\tau(\pi)}}$ est du type `NoOp` (cf. figure 6.11).

La variation d'autonomie dépend de la politique suivie, dans le sens où la probabilité d'avoir une baisse donnée d'autonomie est une fonction de $p^{\zeta(\pi)}$ (probabilité de se retrouver dans une région atteignable). Par exemple, pour atteindre une région r donnée, La baisse d'autonomie sera plus grande avec une politique qui atteint r avec beaucoup de déplacements (p^r faible) qu'avec une politique qui l'atteint avec peu de déplacements (p^r grand). Ainsi, l'arbre de probabilité de la variable autonomie dépend de la variable R' et de la probabilité symbolique $Lp := p^{\zeta(\pi)}$. Cependant, n'ayant pas de modèle réel de la diminution d'autonomie, nous avons considéré une variation d'autonomie statique indépendante de la probabilité des régions atteignables (cf. figure 6.12).

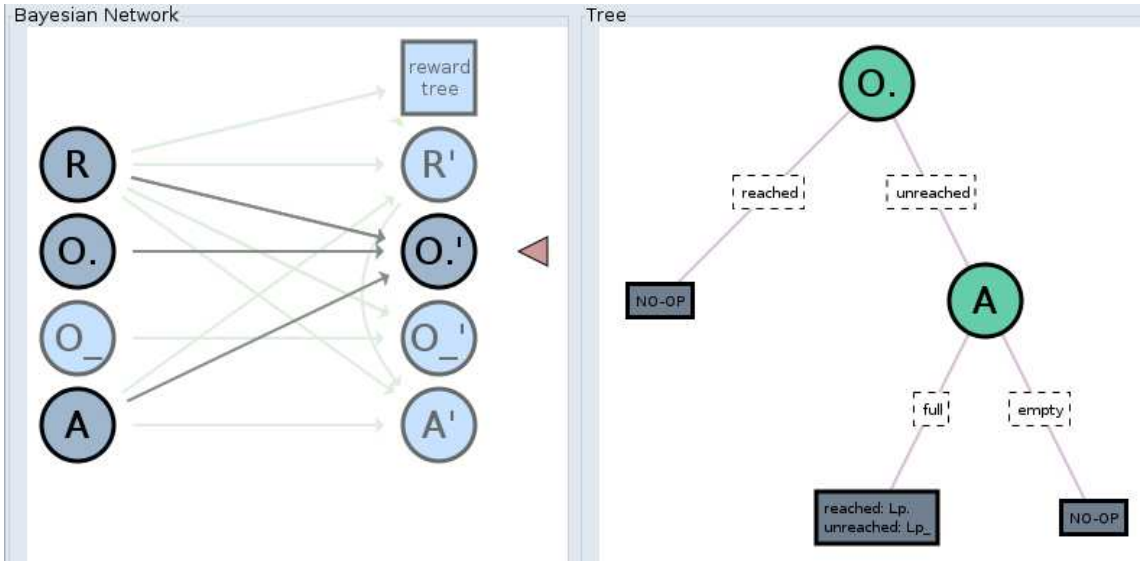


FIG. 6.10 – Capture d'écran de l'éditeur de l'arbre de probabilité générique de la variable symbolique $Y^{\tau(\pi)} = O$.

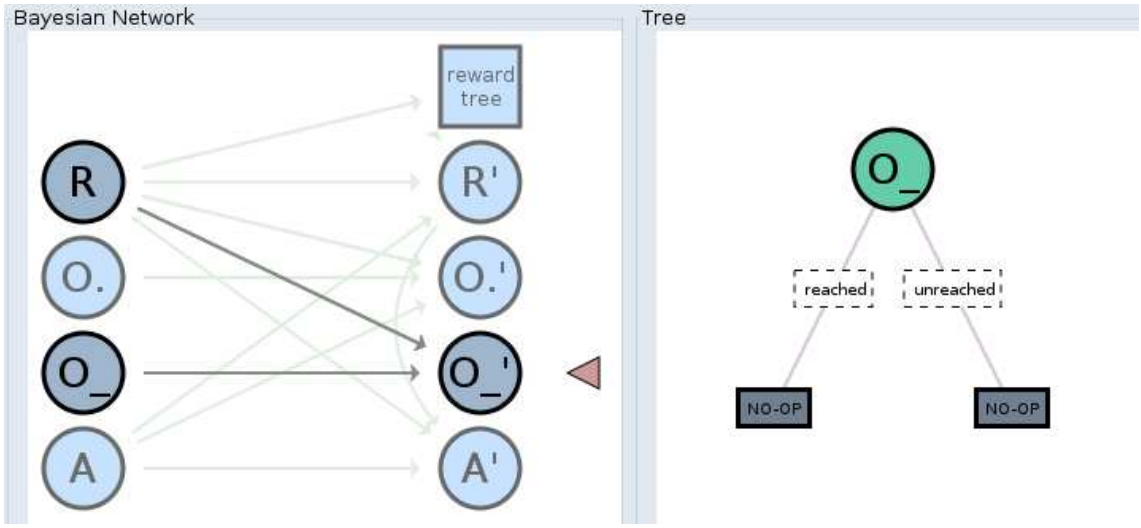


FIG. 6.11 – Capture d'écran de l'éditeur de l'arbre de probabilité générique de la variable symbolique $Y^{\tau(\pi)} = O\_$

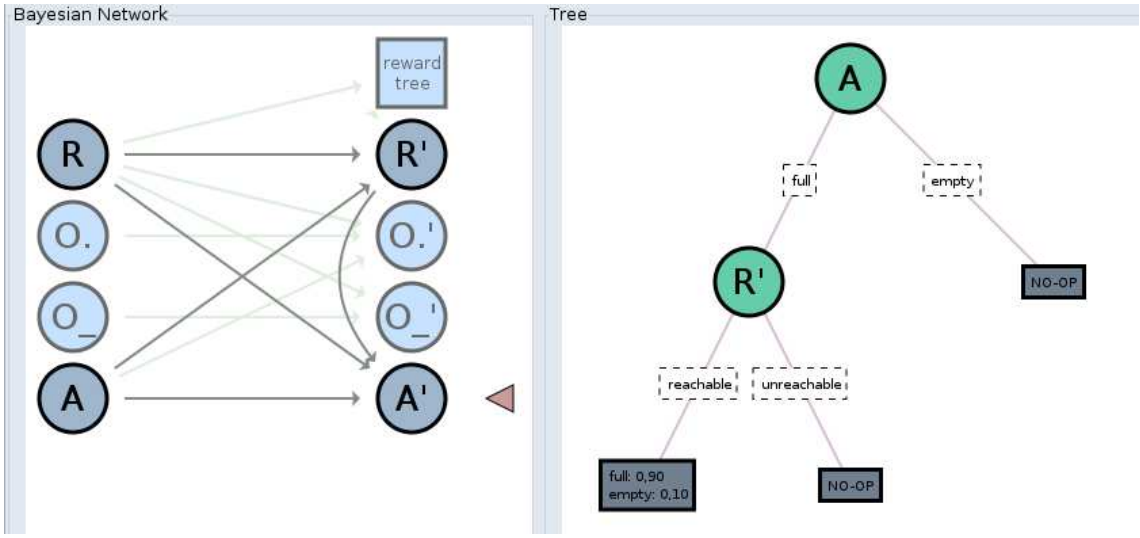


FIG. 6.12 – Capture d'écran de l'éditeur de l'arbre de probabilité de la variable autonomie A

6.2 Instanciation et résolution de la mission

Une fois le sous-espace de navigation et le DBN générique définis, la mission peut être instanciée et résolue à l'aide d'une même vue (cf. figure 6.13). Cette vue représente les options d'optimisation, l'évolution de l'instanciation et de la résolution, ainsi qu'une représentation de la solution (arbre solution abstrait et politiques locales de navigation).

6.2.1 Options d'optimisation

Nous avons choisi de générer le pool de politiques locales à l'aide de techniques de décomposition de MDP énumérés (cf. section 2.1 page 29), car les politiques locales générées dans chaque région sont optimales au regard des récompenses et pénalités qui se trouvent dans les autres régions. Dès qu'une politique locale est générée, nous calculons les macro-probabilités et macro-récompenses associées à cette politique locale, sur la base du théorème 4 (page 36). Lors de l'instanciation automatique du DBN générique, ces macro-transitions instancient les inconnues $Lp.$, $Lv.$, $Lv\_$, Lp : et Lv : (cf. algorithme 19). La figure 6.14 montre l'éditeur des options de l'algorithme de décomposition. Les algorithmes de décomposition sont au choix :

- `LINEAR_PROGRAMMING_DECOMPOSITION` (algorithme de [24], voir page 101)
- `GRID_DECOMPOSITION` (algorithme de [23])

La décomposition du sous-MDP de navigation est assurée par la librairie C++ `graphMDP` (cf. annexe B page 275) que nous avons développé dans le cadre plus

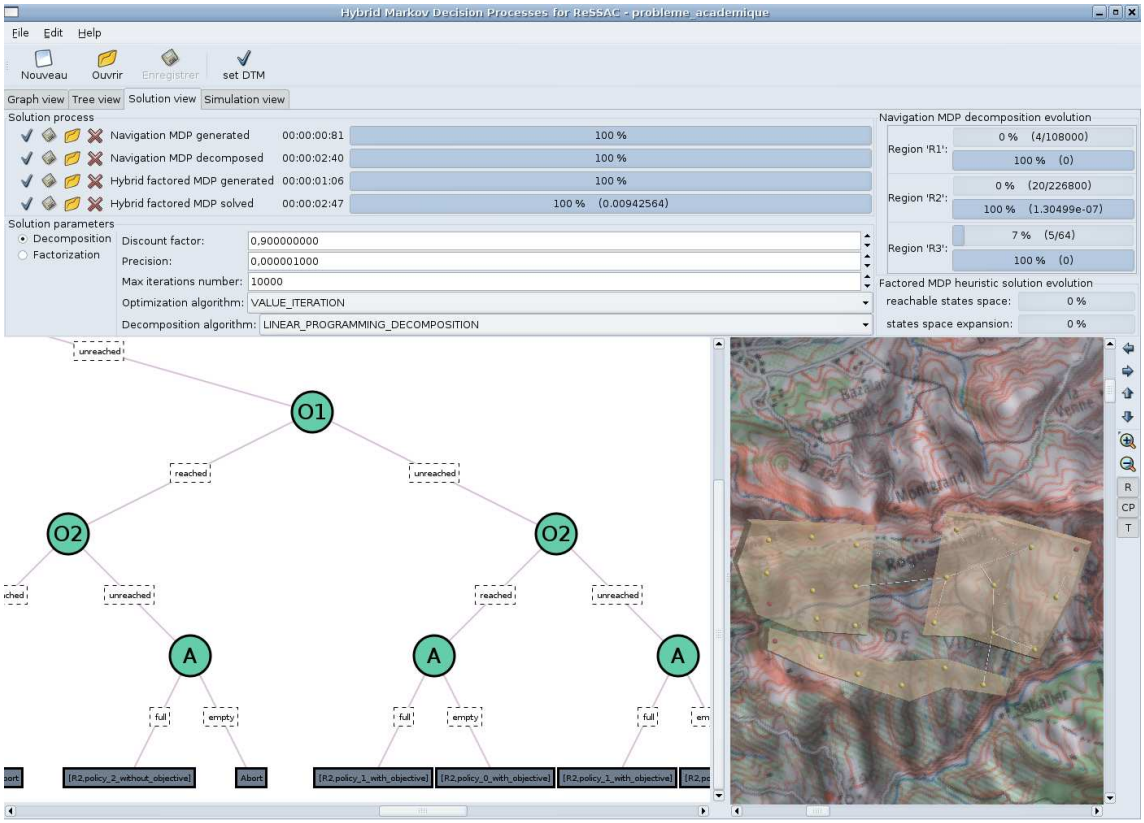


FIG. 6.13 – Capture d'écran de la vue d'instanciation automatique du MDP factorisé abstrait de la mission, et de sa résolution

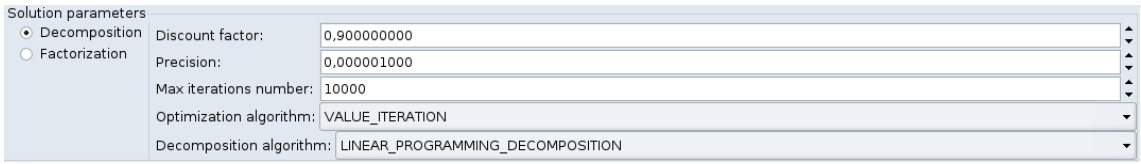


FIG. 6.14 – Capture d'écran de l'éditeur des options de l'algorithme de décomposition

général des MDP énumérés représentés sous forme de graphes [50]. Cette librairie s'appuie sur la librairie d'optimisation linéaire GLPK [61].

Après que les politiques locales aient été générées, le DBN générique est automatiquement instancié afin de définir le MDP factorisé abstrait de la mission. Les options d'optimisation du MDP factorisé sont représentées dans la figure 6.15. Les algorithmes d'optimisation possibles sont (certains seront présentés dans la partie suivante) : `SPUDD/VI`, `SPUDD/PI`, `PsfDP/VI`, `PsfDP/PI`, `AsfDP/VI`, `AsfDP/PI`, `IPsfDP/VI`, `IPsfDP/PI`, `IAsfDP/VI`, `IAsfDP/PI`.

FIG. 6.15 – Capture d'écran de l'éditeur des options de l'algorithme d'optimisation du MDP factorisé abstrait

FIG. 6.16 – Capture d'écran du suivi du processus d'instanciation et de résolution

La conversion automatique des arbres de décision en ADD ainsi que la résolution du MDP factorisé est réalisée par la librairie C++ `treeMDP` que nous avons développé dans le cadre général des MDP factorisés (cf. annexe C). Les ADD sont gérés par la librairie CUDD [70].

6.2.2 Suivi du processus d'instanciation et de résolution

L'interface permet de suivre l'évolution du processus d'instanciation du DBN générique, dont la génération des politiques locales, et de la résolution du MDP factorisé abstrait (cf. figure 6.16). En ce qui concerne la construction du MDP énuméré représentant le sous-espace de navigation, le nombre d'états et de transitions à créer est connu si bien que la barre de progression indique le pourcentage d'éléments construits. De même, dès que le pool de politiques locales est généré, le nombre de DBN et d'arbres de décision à construire est connu, si bien que la barre de progression de génération du MDP factorisé abstrait indique le pourcentage d'arbres de décision instanciés.

En revanche, le nombre de politiques locales générées ainsi que le nombre d'itérations de l'algorithme d'optimisation du MDP factorisé ne sont pas connus à l'avance. En revanche, ces deux algorithmes s'arrêtent lorsque l'erreur de Bellman devient inférieure à ϵ [24, 14]. Ainsi, dans ces deux cas, la barre de progression indique l'opposé de l'erreur relative b entre l'erreur de Bellman courante e et la précision ϵ :

$$b = \begin{cases} 100 & \text{si } e < \epsilon \\ E_{\mathbb{N}}\left(\frac{e_0 - e}{e_0 - \epsilon}\right) \times 100 & \text{sinon} \end{cases}$$

L'évolution de la décomposition du sous-MDP de navigation est représenté dans la figure 6.17. Pour chaque région r, la première barre de progression indique le pourcentage de politiques locales générées par rapport au nombre total de politiques

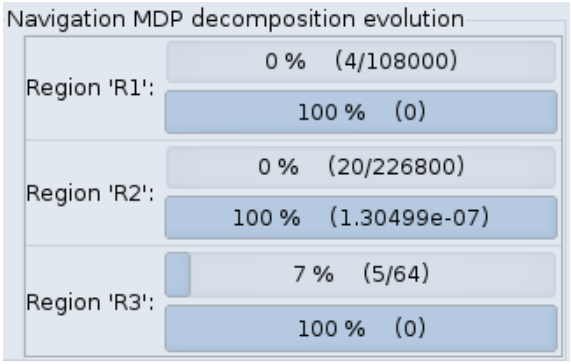


FIG. 6.17 – Capture d'écran du suivi du processus de génération des politiques locales

locales possibles dans cette région ($|\mathcal{A}|^{|r|}$). La deuxième barre de progression représente l'opposé de l'erreur relative entre la plus grande erreur de Bellman pour la politique qui vient d'être générées, et la précision [24]. Cet exemple, aussi simple qu'il soit, montre que le nombre de politiques locales générées par la méthode de [24] est très faible. Nous verrons plus loin que cette remarque sera vérifiée sur des problèmes beaucoup plus difficiles.

Enfin, les processus d'instanciation et de résolution sont lancés dans des threads propres, ce qui permet de les arrêter à tout moment. Ceci est particulièrement utile lors de la résolution de problèmes difficiles dont l'optimisation peut durer plusieurs heures, ou saturer la mémoire de la machine.

6.2.3 Représentation de la solution

La solution est constituée d'une part de l'arbre de solution du MDP abstrait dont les feuilles sont des politiques locales de navigation, et d'une représentation 3D des politiques locales de chaque feuille d'autre part (cf. figure 6.18). En sélectionnant une feuille de l'arbre solution, les transitions de la politique locale sont directement représentées.

Les figures 6.18 et 6.19 montrent que les variables symboliques $O^{\tau(\pi)}$ et $O^{\overline{\tau(\pi)}}$, ainsi que les sous-arbres symboliques de la variable R, ont été instanciés.

Deux types de feuille sont visibles sur l'arbre solution de la figure 6.18 :

- `Abort` : action spéciale cachée à l'utilisateur, qui est appliquée lorsqu'aucune autre action n'est applicable (union des ADD vide) ou lorsque toutes les autres actions sont équivalentes ; ceci arrive par exemple lorsque tous les objectifs de la mission ont été atteints, puisque l'environnement ne contient plus de récompense dans ce cas.
- `R3_policy_0_with_objective` : nom de la politique locale à appliquer, lue

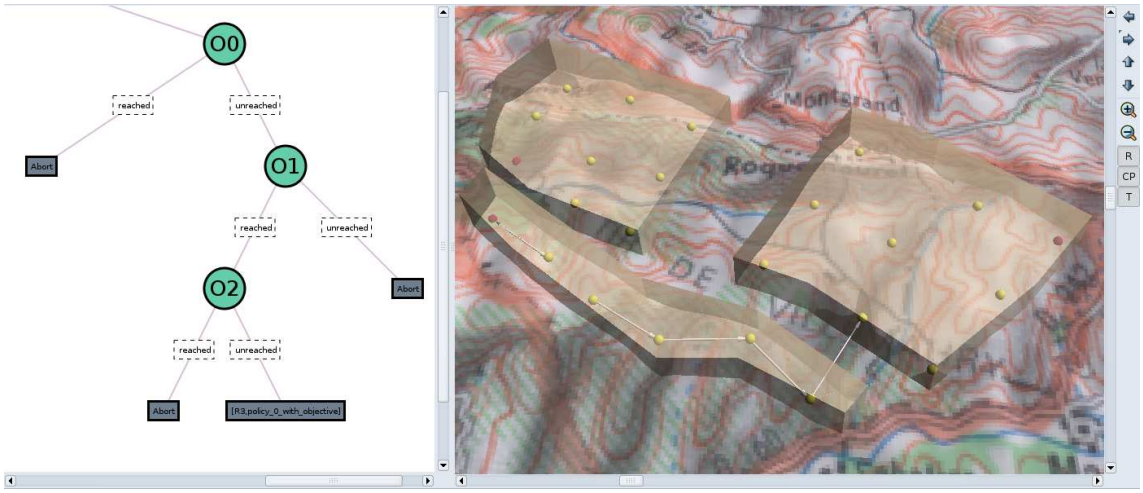


FIG. 6.18 – Capture d'écran de la représentation graphique de la solution

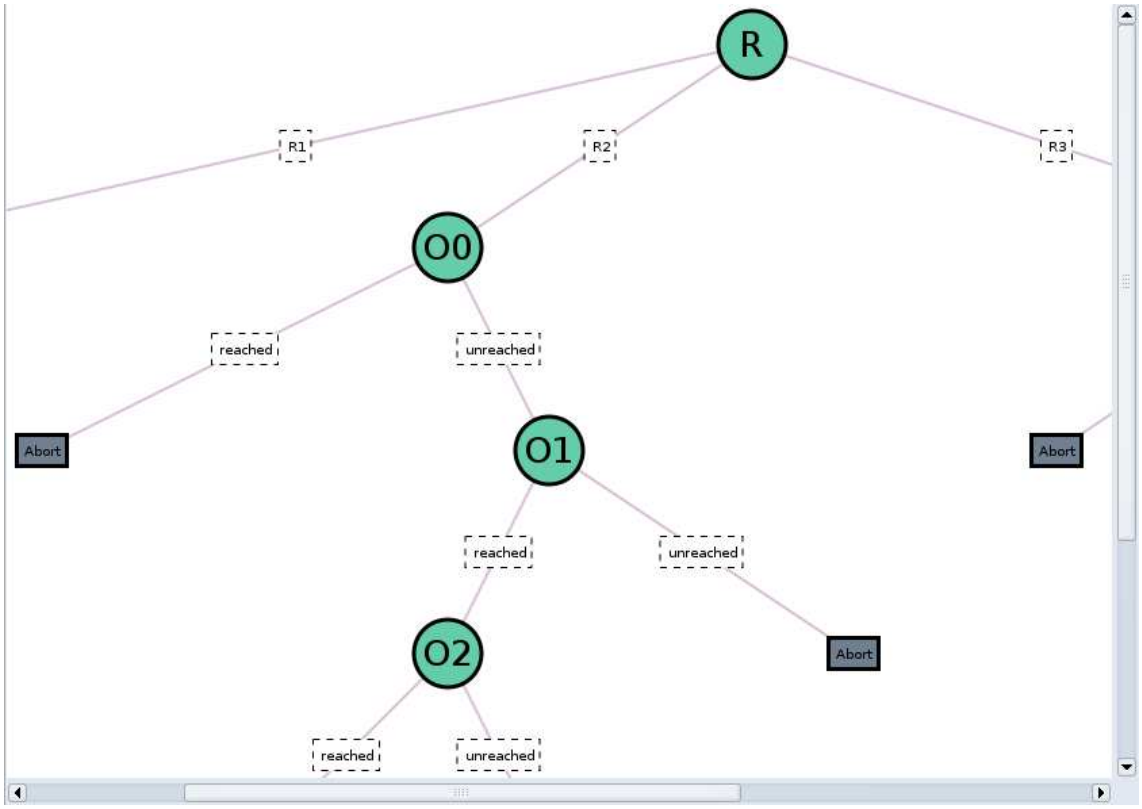


FIG. 6.19 – Capture d'écran d'une vue partielle de l'arbre solution du MDP factorisé abstrait

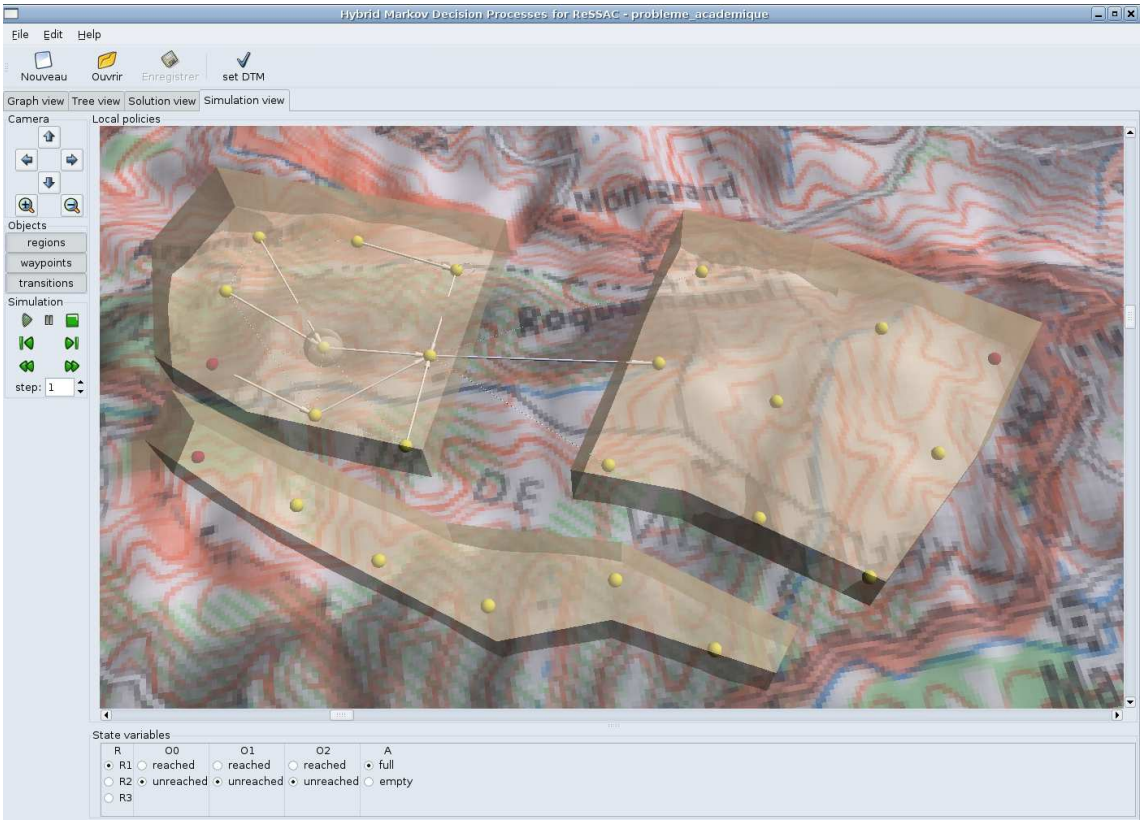


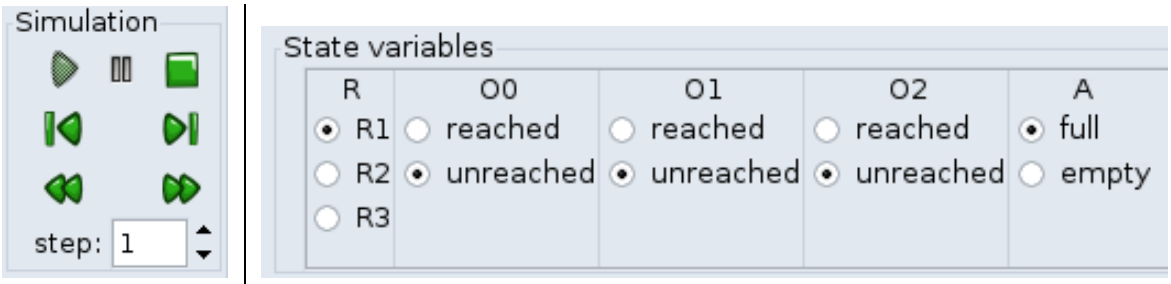
FIG. 6.20 – Capture d'écran de la vue de simulation de la solution de la mission

par le parser, qui se décompose ainsi :
- `R3` : région où la politique est définie
- `policy_0` : numéro de la politique locale définie dans la région `R3`
- `with_objective` : classe de la politique locale (indique ici que l'objectif est encore dans la région et qu'il n'a donc pas encore été atteint).

6.2.4 Simulation de la solution

Nous avons implémenté une vue de simulation qui permet de jouer la politique globale obtenue (cf. figure 6.20). En effet, dans le cas de problèmes de grande taille, la politique globale peut être difficile à lire. Des simulations sont alors plus appropriées pour vérifier et comprendre la politique globale. Elles permettent également de valider notre approche hiérarchique et générique.

L'avancement de la simulation peut être contrôlé à tout moment (cf. figure 6.21.a). En particulier, l'utilisateur peut revenir en arrière afin d'essayer de jouer des transitions stochastiques différentes de celles qui ont été aléatoirement choisies.



(a) (b)

FIG. 6.21 – Capture d'écran du contrôle de la simulation (a) et du suivi valeurs des variables d'état (b)

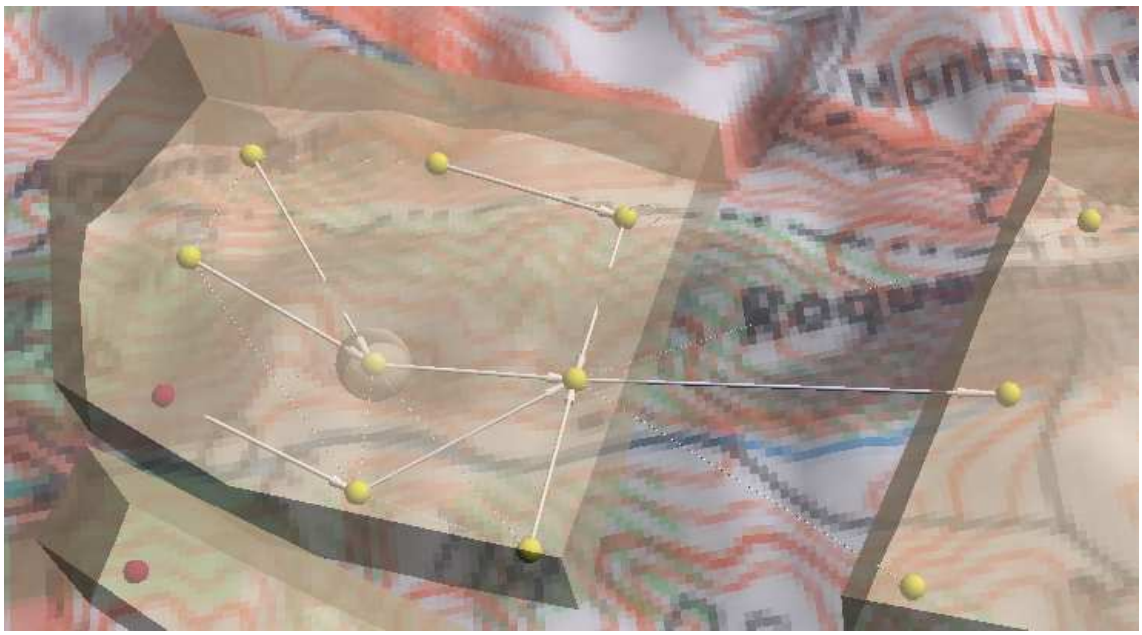


FIG. 6.22 – Capture d'écran de la simulation de la politique locale courante

Une vue (cf. figure 6.21.b) représente les valeurs courantes des variables d'état.

Une sphère représente le point de passage courant durant la simulation (cf. figure 6.22). Les transitions de navigation sont tirées aléatoirement dans le modèle de transitions du MDP énuméré de navigation.

La politique locale courante termine lorsque :

- le point de passage courant sort de la région où est définie la politique locale courante,
- *ou* l'objectif interne de cette région est atteint

Dès qu'une politique locale se termine, les valeurs des variables d'état sont mises

à jour aléatoirement d'après les arbres de décision *instanciées* de la politique qui vient de se terminer. A cause des dépendances synchrones (A' dépend de R' par exemple), une analyse de l'ordre de tirage aléatoire des variables d'état post-action doit être effectuée préalablement (voir [29] à ce sujet).

C'est sur ce dernier point que notre simulation se différencie du cas réel. En effet, nous supposons durant la simulation que les variables d'état autres que R ne varient pas pendant que la politique locale n'est pas terminée. Ceci rejoint l'hypothèse de découplage du chapitre 3 (page 65) sur la réduction automatique d'arité d'une variable d'état quelconque. Dans le cas réel, toutes les variables d'état sont susceptibles de changer a cours de chaque trajectoire, ce qui nécessiterait de mettre à jour l'ensemble des variables d'état après chaque trajectoire. Ceci est impossible dans notre simulation puisque la dynamique de notre modèle générique de DBN est définie au niveau des régions et non des points de passage.

6.3 Autres problèmes testés

Nous avons testé notre approche hiérarchique et générique sur des missions de types et de tailles différents. Les missions de très grande taille n'ont pas encore été modélisées à l'aide de l'outil de modélisation de mission car le sous-espace de navigation est très grand. Dans le futur, le sous-espace de navigation sera généré par un logiciel externe qui génère automatiquement les points de passage et les trajectoires de vol à partir d'un modèle numérique de terrain [87]. Ainsi, les missions de grande taille ont été automatiquement générées sous forme de «grilles» de navigation fréquemment rencontrées dans la littérature MDP. En revanche, les variables d'état et le DBN générique utilisés sont identiques à ceux de la section précédente.

6.3.1 Missions «linéaires»

Les missions «linéaires» sont constituées d'une liste de régions "en enfilade" : chaque région est connectée à la région qui la précède dans la liste, et à celle qui lui succède (cf. figure 6.23). Elles ont la caractéristique d'avoir un grand nombre de régions, et donc un grand nombre de variables d'état (1 objectif par région), mais le sous-problème de navigation est simple à résoudre. En effet, la connexion entre les régions est très faible, ce qui simplifie la décomposition des MDP locaux.

Dans la première série de tests, nous avons appliqué notre méthode à des missions linéaires de tailles différentes (cf. tableau 6.1). Les politiques locales ont générées avec l'algorithme de [24], et le MDP factorisé abstrait a été optimisé à l'aide de l'algorithme SPUDD [88].

Il apparaît que le nombre de politiques locales générées reste peu important et que le temps de décomposition est toujours faible, même pour des problèmes de très

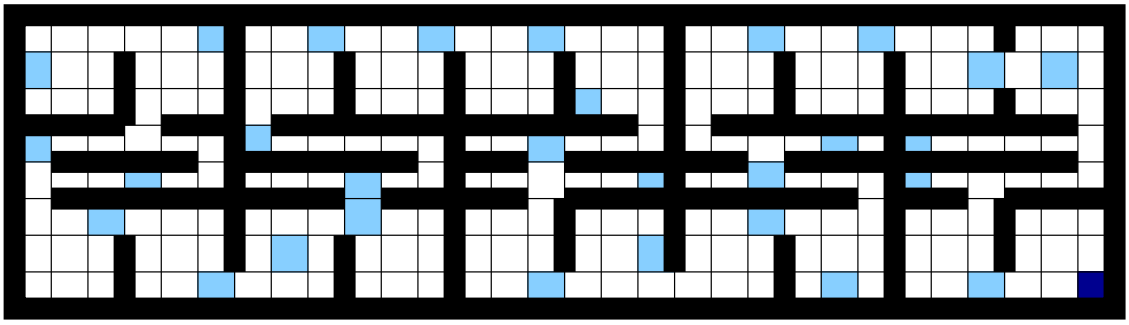


FIG. 6.23 – Mission «linéaire» : 30 buts, 1 variable binaire supplémentaire d'autonomie, grille 8 × 30, 515396075520 états énumérés

Nb états énumérés	Nb régions	Nb politiques locales générées	Tps décomposition	Tps instanciation	Tps optimisation
384	3	10	0,08	< 0,01	0,02
6144	6	33	0,38	0,01	1,51
73728	9	47	0,56	0,02	26,8
786432	12	59	0,63	0,02	192,1
1720320	13	65	0,67	0,04	467,11
7864320	15	73	0,79	0,05	4344,02
515396075520	30	156	2,19	0,25	—

TAB. 6.1 – Comparaison des temps de décomposition, instanciation et optimisation du MDP hiérarchique pour des missions «linéaires» (en secondes avec un processeur P4 de 2,8 GHz)

grande taille ($\approx 10^{12}$ états), ce qui montre que la composante de navigation des missions linéaires est facile à résoudre.

De plus, le temps d'instanciation reste négligeable devant le temps total de résolution. Ce résultat était attendu, étant donné que la complexité de notre algorithme d'instanciation automatique du DBN générique est linéaire en la taille des arbres du DBN générique (généralement très petite) et en le nombre de politiques locales générées (faible grâce à l'algorithme de [24]). A titre de comparaison, les algorithmes de décomposition et d'optimisation du MDP factorisé sont polynomiaux en le cardinal de leur espace d'états, qui peut être potentiellement énorme.

Enfin, nous remarquons que l'algorithme d'optimisation exacte `SPUDD` ne peut pas résoudre le MDP factorisé pour la mission de plus grande taille. Dans la partie suivante, nous proposerons un algorithme qui permet de résoudre approximativement la mission sur un sous-espace d'états atteignable à partir du point de passage initial.

Par ailleurs, nous avons comparé la résolution de la mission (modélisation et optimisation) pour un modèle énuméré classique de MDP et pour notre modèle hiérar-

Nb états énumérés	Nb régions	modèle	Tps construction	Tps décomposition	Nb politiques locales générées	Tps optimisation
384	3	factorisé	< 0,01	0,08	10	0,02
		énuméré	0,03	–	–	0,01
6144	6	factorisé	0,01	0,38	33	1,51
		énuméré	4,21	–	–	0,38
73728	9	factorisé	0,03	0,56	47	26,8
		énuméré	587,62	–	–	5,03

TAB. 6.2 – Comparaison entre les approches à base de MDP abstrait factorisé et de MDP énuméré pour des missions «linéaires» (en secondes avec un processeur P4 de 2,8 GHz)

chique générique (cf. tableau 6.2). Remarquons d'abord que les différentes missions n'auraient pas pu être modélisées sous forme de MDP factorisé non hiérarchique, c'est-à-dire avec une variable de navigation microscopique. En effet, la plus grande mission «linéaire» testée a 240 points de passage, ce qui aurait conduit à une variable de navigation avec 240 valeurs possibles, au point que la modélisation manuelle des arbres de décision aurait été rédhibitoire (la représentation n'est pas adaptée au domaine). Dans le cas d'une modélisation complètement énumérée, le sous-espace de navigation peut être relativement simple à modéliser à l'aide d'un outil de modélisation 3D ; le sous-espace de navigation doit ensuite être orthogonalement multiplié par les autres variables d'état du problème. Cette approche n'est pas forcément plus simple, mais c'est celle que nous avons comparé à notre modèle hiérarchique.

Le tableau 6.2 montre que l'optimisation du modèle énuméré est certes plus rapide que celle du modèle factorisé, mais la construction du modèle est beaucoup plus longue. La construction du modèle énuméré comprend la multiplication du sous-espace de navigation par les autres variables d'état. Celle du modèle factorisé consiste à instancier le DBN générique avec les politiques locales générées par décomposition du sous-espace de navigation. L'optimisation du modèle énuméré est performante car les structures informatiques et les algorithmes utilisées sont beaucoup plus simples que celles du modèle factorisé [5]. Malheureusement, le temps gagné sur l'optimisation est très largement perdu sur le temps passé à construire le MDP de la mission. Ces résultats, qui seront confirmés avec les test sur des missions «concentriques», montrent donc l'intérêt de notre approche «diviser pour régner».

6.3.2 Missions «concentriques»

Les missions «concentriques» sont constituées de régions fortement connectées autour d'une région centrale (cf. figure 6.24). La majorité des régions sont connectées à 3 ou 4 autres régions, ce qui complique la décomposition du sous-MDP de navigation, dont la complexité est exponentielle en le nombre de sorties des régions [23, 24]. De plus, certaines missions ont été modélisées avec 81 états par régions,

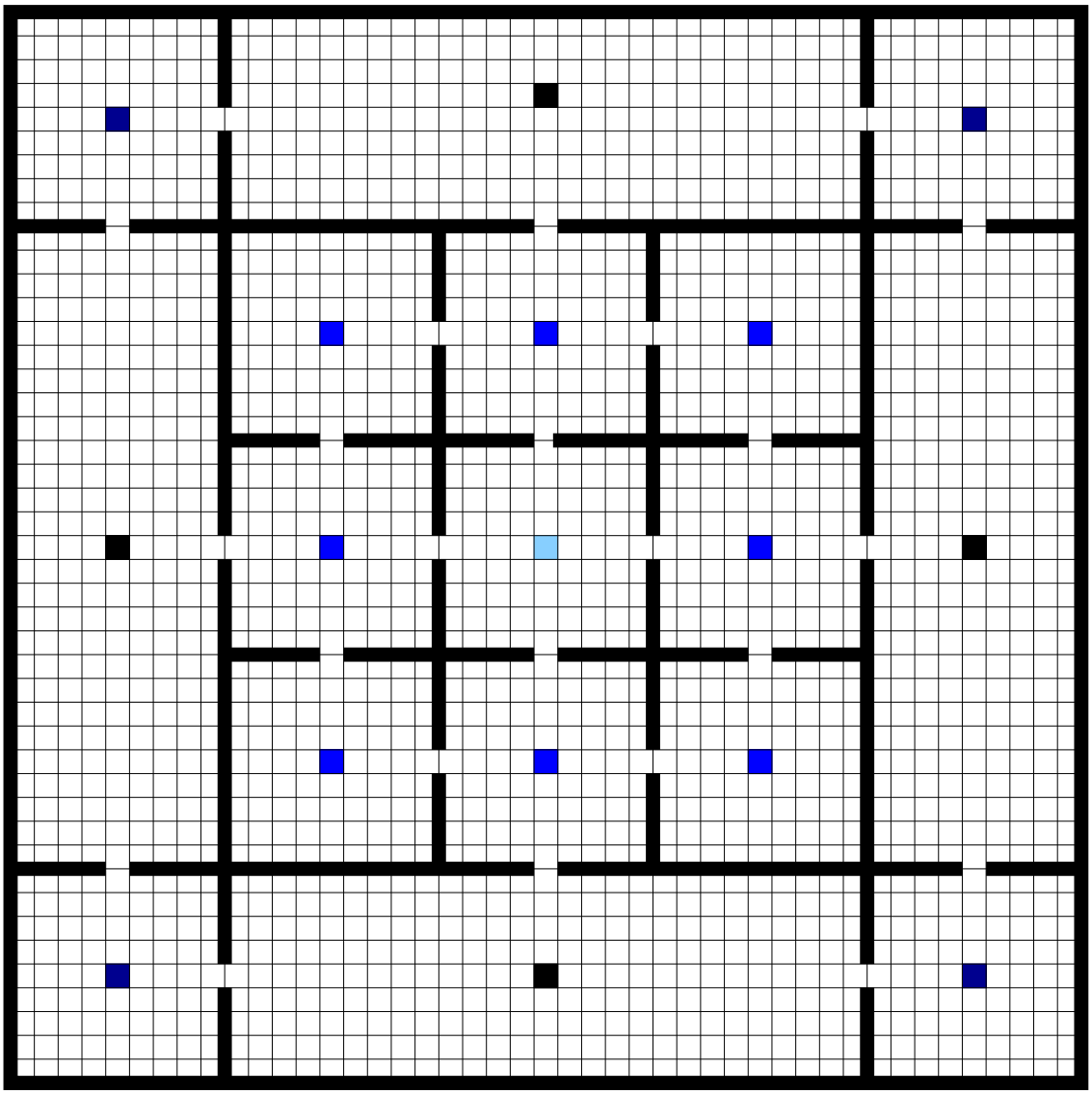


FIG. 6.24 – Mission «concentrique» : 17 buts, 1 variable binaire supplémentaire d'autonomie, grille 45 × 45, 530841600 états énumérés

contrairement à un maximum de 9 états par région pour les missions linéaires. Ceci complexifie également la décomposition du sous-MDP de navigation car le nombre de politiques locales par région r est exponentiel en le nombre de ses états ($|\mathcal{A}|^{|r|}$). Nous avons réalisé les deux mêmes séries de tests que ceux effectué avec les modèles linéaires.

Le tableau 6.3 montre que le temps d'instanciation est négligeable devant le temps de résolution global. De plus, les problèmes linéraires sont plus difficiles à dé-

Nb états énumérés	Nb régions (états/région)	Nb politiques locales générées	Tps décomposition	Tps instanciation	Tps optimisation
82944	9 (9)	21	0,13	0,01	0,12
746496	9 (81)	61	40,61	0,01	16,77
58982400	17 (9)	69	0,58	0,03	1621,62
530841600	17 (81)	117	93,83	0,06	—

TAB. 6.3 – Comparaison des temps de décomposition, instanciation et optimisation du MDP hiérarchique pour des missions «concentriques» (en secondes avec un processeur P4 de 2,8 GHz)

composer que les problèmes linéaires, comme cela était prévu. En revanche, à nombre d'états énumérés équivalent, l'optimisation du MDP factorisé est plus simple pour des missions concentriques (1621,62 s. pour 58982400 états énumérés en concentrique, contre 4344,2 s. pour 7864320 états énumérés en linéaire). En effet, la plus grande complexité du sous-espace de navigation est compensée par un nombre bien plus faible de variables d'état, qui est le facteur déterminant de la complexité de l'algorithme d'optimisation des MDP factorisés [26, 31].

Ces résultats montrent la prépondérance de l'optimisation du MDP factorisé sur la décomposition du sous-MDP de navigation, et donc l'intérêt de notre modèle hiérarchique. Il est primordial de réduire au maximum le nombre de variables d'état de la mission, ce qui est équivalent à réduire au maximum l'arité des variables d'état. Une modélisation complètement factorisée de la mission concentrique la plus difficile (2025 points de passage) aurait nécessiter $E_{\mathbb{N}}(\log_2(2025)+1) = 11$ variables binaires `point de passage` au lieu de $E_{\mathbb{N}}(\log_2(17)) + 1 = 5$ variables binaires `région`, soit une augmentation non négligeable de 26% du nombre total de variables binaires !

La comparaison entre les modèles énumérés et factorisés des missions concentriques (cf. tableau 6.4) confirme en partie les résultats obtenus pour les missions linéaires. Le temps de construction est bien plus faible en modélisation factorisée, ce qui confirme la force de modélisation de notre modèle hiérarchique. D'ailleurs, le plus grand problème testé ne peut même pas être modélisé en énuméré. En revanche, contrairement aux missions linéaires, l'optimisation du MDP factorisé abstrait est nettement plus rapide que celle du MDP énuméré classique. En effet, dans le cas des missions concentriques, la composante de navigation est la plus difficile à résoudre, mais elle est prise en charge par l'algorithme de décomposition du sous-MDP de navigation. Ainsi, l'optimisation du MDP classique passe du temps à résoudre la composante prépondérante de navigation, ce dont est déchargé l'optimisation du MDP factorisé abstrait. Ceci n'est pas le cas des missions linéaires, dont la composante de navigation est, au contraire, négligeable devant l'ensemble des composantes de la mission.

Nb états énumérés	Nb régions (états/région)	modèle	Tps construction	Tps décomposition	Nb politiques locales générées	Tps optimisation
82944	9 (9)	factorisé	0,02	0,13	21	0,12
		énuméré	746,98	–	–	2,25
746496	9 (81)	factorisé	0,02	40,61	61	16,77
		énuméré	> 1hr	–	–	–

TAB. 6.4 – Comparaison entre les approches à base de MDP abstrait factorisé et de MDP énuméré pour des missions «concentriques» (en secondes avec un processeur P4 de 2,8 GHz)

Bilan

Nous proposons un modèle générique hiérarchique permettant de simplifier la phase de modélisation d'un problème de planification stochastique. De nombreux problèmes sont naturellement factorisés en composantes diverses, au point que des modèles factorisés de MDP ont été récemment proposés pour résoudre ces problèmes. Cependant, certaines variables d'état ont parfois un grand nombre de valeurs possibles, comme par exemple la composante de navigation d'une mission ReSSAC. La modélisation factorisée du MDP est alors très difficile, car les arbres de décision sont très larges et illisibles.

Nous proposons de définir le MDP à un niveau plus abstrait, où une variable de grande arité est partitionnée en une variable abstraite de plus faible arité. Les actions de ce MDP hiérarchique sont des politiques locales définies chacune sur une partition du sous-espace engendré par les valeurs de la variable abstraite. Néanmoins, les politiques locales sont potentiellement très nombreuses, si bien que nous avons introduit la notion de DBN générique, qui représente symboliquement tous les DBN du MDP factorisé hiérarchique. Notre DBN générique est paramétré par une classe quelconque de politiques locales et la partition où les politiques locales de cette classe s'appliquent.

Nous avons montré la facilité d'utilisation et la pertinence de notre modèle générique et hiérarchique à travers un outil de modélisation et de résolution de missions ReSSAC. La performance de notre approche a été évaluée sur des problèmes d'exploration de grandes tailles. Notre approche hiérarchique permet de construire des modèles de mission de manière quasiment instantanée, alors que les modèles énumérés classiques correspondants ne pouvaient même pas être construits pour les problèmes les plus difficiles. Des comparaisons avec des modèles équivalents complètement factorisés restent à faire, mais nous avons montré toutefois que le coût d'optimisation du MDP factorisé est prépondérant sur le coût de réduction de l'arité d'une variable d'état. Ainsi, un modèle complètement factorisé nécessiterait un accroissement non négligeable de variables d'état par rapport à notre modèle hiérarchique, si bien que sa résolution globale *devrait* être plus difficile. Notre approche hiérarchique est également d'autant plus performante que la composante prépondérante de la mission correspond à la variable réduite.

Cependant, notre modèle hiérarchique suppose que le problème de planification soit défini *par essence* au niveau microscopique, afin qu'il n'y ait aucune raison de distinguer sémantiquement les partitions du sous-espace engendré par la variable d'état réduite. Sous cette hypothèse, seule la partition où s'applique la politique locale de notre DBN générique, ainsi que les partitions atteignables par cette politique locale, peuvent être raisonnablement distinguées des autres régions. Notre modèle générique ne s'applique pas si cette hypothèse n'est pas respectée.

Considérons par exemple une mission ReSSAC où la communication radio entre la station sol et l'hélicoptère autonome ne peut être établie que dans une seule région de l'environnement, uniformément sur tous les points de passage de cette région. La probabilité d'établir une communication radio entre la station sol et le drone dépend donc de la région où se trouve le drone par rapport à **la région spécifique** où la communication est possible. Cette région doit donc être distinguée de la région où s'applique la politique locale paramètre du DBN générique. Malheureusement, notre modèle générique et hiérarchique ne permet pas de prendre en compte une partition spécifique du sous-espace engendré par la variable réduite. Ainsi, le traitement statique de partitions spécifiques serait sans doute une perspective naturelle de ce travail.

Enfin, concluons ce chapitre en remarquant que nos tests ont montré l'incapacité des algorithmes optimaux et globaux d'optimisation des MDP factorisés (itération de la valeur ou de la politique) à résoudre des problèmes d'exploration de grande taille. Dans le chapitre suivant, nous présenterons une classe d'algorithmes symboliques et heuristiques que nous avons développé afin de résoudre des problèmes de planification stochastique de grande taille sur un sous-espace d'états atteignables depuis un ensemble d'états initiaux donnés.

Quatrième partie

`sfDP` : une classe d'algorithmes de Programmation Dynamique Symbolique, Heuristique et Focalisée

Motivation

Dans le chapitre précédent, nous avons effectué des tests qui montrent que les algorithmes de résolution optimale des MDP factorisés tels SPI [27] ou SPUDD [32] sont incapables de résoudre des problèmes de planification stochastique de très grande taille. La représentation de la fonction de valeur sous forme d'arbre de décision ou d'ADD est compacte à condition que ces structures restent creuses. Néanmoins, au cours de la résolution d'un MDP, la fonction de valeur se structure de plus en plus, en ce sens que le nombre de sous-ensembles d'états où la fonction de valeur est constante sont de plus en plus nombreux. Ainsi, la taille des arbres de décision ou des ADD augmente sans cesse, puisque les feuilles représentent des sous-ensembles d'états de même valeur.

Des algorithmes [28, 89, 55, 33] ont été proposés afin de limiter le nombre de sous-ensembles d'états de même valeurs. Ils reposent sur une approximation de la fonction de valeur qui relâche la structure de la fonction de valeur : ceci a pour effet de réduire le nombre de feuilles des arbres de décision ou des ADDs. D'autres travaux [90] portent sur la résolution des MDP factorisés avec l'algorithme d'itération de la politique, car la politique est plus structurée que sa valeur. Dans le cas des MDP factorisés, la politique est plus difficile à gérer que la fonction de valeur car l'accès aux feuilles de l'arbre de politique pour un état donné est coûteux. La solution consiste à représenter la politique sous forme d'une liste d'arbres binaires (ou de BDD) qui représentent pour chaque action l'ensemble des états sur lequel l'action s'applique.

Si ces algorithmes permettent de réduire la complexité de l'algorithme d'itération de la valeur structurée, ils n'échappent cependant pas à l'explosion combinatoire qui survient avec un grand nombre de variables d'état. Ils permettent seulement de retarder quelque peu l'effet de cette explosion combinatoire. Ainsi, un algorithme récent [91] a été proposé pour résoudre des MDP factorisés sur un sous-ensemble d'états atteignables depuis un état initial donné. Il s'agit de `sLAO`*, une version symbolique de `LAO`* [92]. Ce dernier algorithme est une variante de `AO`* [93] adaptée au graphes cycliques ET/OU. L'algorithme `sLAO`*s'avère être très performant pour des problèmes où l'état de départ est connu, car l'espace d'états exploré est généralement très faible comparé à l'espace d'états développé par les algorithmes optimaux

ou approximatifs qui optimisent le MDP sur tout l'espace d'états. Cet algorithme est donc très utile pour des missions d'exploration du type ReSSAC où l'état géographique ainsi que les valeurs de toutes les autres variables d'état sont connus au début de la mission.

Néanmoins, certains problèmes incluent également des sous-buts à atteindre, comme par exemple les objectifs internes aux régions d'une mission ReSSAc. L'algorithme sLAO*, guidé comme nous le verrons par la zone atteignable de plus grande récompense, est parfaitement adapté à un problème comprenant un seul but à atteindre : il suffit que la seule récompense du modèle soit donnée à ce sous-but. Malheureusement, si on ajoute des sous-buts au problème, sLAO*restera attiré par la zone de plus forte récompense. au mieux, il explorera trop d'états ; au pire, la politique optimale ne rejoindra pas les sous-buts imposés.

Par conséquent, nous proposons dans ce chapitre un algorithme qui focalise la recherche des états atteignables sur une liste imposée de sous-buts à atteindre, dont la priorité est d'atteindre ces sous-buts. Une fois cette condition remplie, notre algorithme optimise la politique, qui consiste à atteindre ces sous-buts, sur l'espace d'états atteignables courant. Ainsi, la différence fondamentale entre sLAO*et notre algorithme repose sur l'espace d'états atteignables.

Nous commencerons cette dernière partie avec une présentation de sLAO*. Nous proposerons ensuite notre algorithme intitulé sfDP (Symbolic Focused Dynamic Programming) qui, comme nous le verrons, se décline en plusieurs versions. Nous terminerons cette partie par des tests qui montrent l'intérêt de notre algorithme pour des problèmes dont des états initiaux possibles et les positions des récompenses sont connus.

Plan de la partie IV

Chapitre 7 : présentation de sLAO* : étude théorique et algorithme

Chapitre 8 : proposition d'une nouvelle heuristique symbolique qui guide la recherche des solutions du MDP vers les chemins menant aux récompenses avec une probabilité maximale

Chapitre 9 : proposition d'un nouvel algorithme symbolique de planification stochastique, qui utilise une heuristique focalisée sur les récompenses du MDP

Chapitre 7

sLAO* : un algorithme de programmation dynamique symbolique et heuristique

7.1 De Dijkstra à sLAO*

La recherche du plus court chemin dans les graphes orientés est fondamentale en Intelligence Artificielle, tant ses applications sont nombreuses. En planification, elle permet de guider la recherche vers des zones d'intérêt. Le guidage de la solution grâce à des informations externes concrètes (la distance directe à un but par exemple) s'appelle l'*heuristique*. La solution obtenue par un algorithme heuristique n'est pas nécessairement optimale : la solution est d'autant mieux estimée que l'heuristique est bonne.

7.1.1 Graphe «OU» et planification déterministe

L'algorithme de Dijkstra [8] trouve le plus court chemin depuis un état initial donné et tous les autres états, dans un graphe orienté à poids positifs. Si les poids sont négatifs, l'algorithme de Bellman-Ford [94] résout le même problème mais à une performance moindre. De tels graphes peuvent être considérés comme des graphes «OU» : à chaque nœud graphe, plusieurs chemins exclusifs sont possibles. Le but est de trouver l'arrête optimale à chaque nœud *admissible*, c'est-à-dire qui appartient à un chemin provenant de l'état initial. Ce formalisme s'applique parfaitement à la planification de chemin déterministe [12], en considérant que les nœds sont les états de l'environnement et que chaque arrête d'un nœud correspond à une action applicable à ce nœud (cf. figure 7.1).

Lorsqu'une heuristique est disponible, il est intéressant de l'utiliser afin de guider

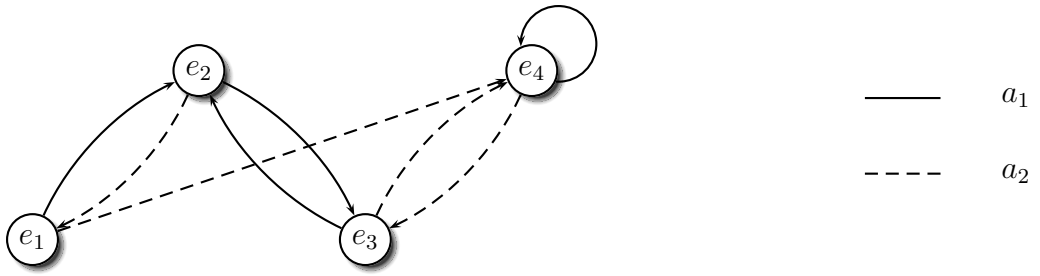


FIG. 7.1 – Exemple de graphe de planification «OU» : 4 états et 2 actions déterministes

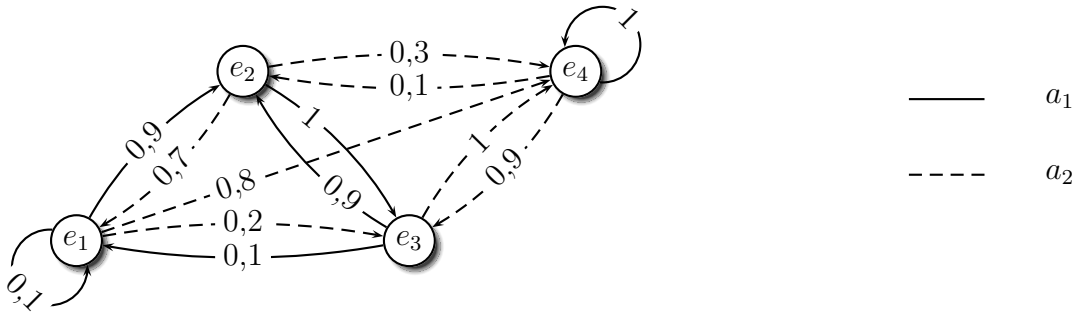


FIG. 7.2 – Exemple de graphe de planification «ET/OU» : 4 états et 2 actions stochastiques

la recherche de la solution. Par exemple, dans les problèmes de plus court chemin, il est souvent possible de disposer de la distance de n'importe quel état l'état de départ. Ainsi, l'algorithme A* [9] et ses nombreuses variantes utilisent une heuristique donnée afin de n'explorer que les nœuds du graphe les plus intéressants au regard de cette heuristique. A condition que l'heuristique ne surestime jamais les coûts, A* est *admissible*, c'est-à-dire qu'il trouve toujours le plus court chemin depuis l'état initial.

7.1.2 Graphe «ET/OU» et planification stochastique

Néanmoins, certains problèmes nécessitent une modélisation sous forme de graphes abstraits «ET/OU» tels que chaque arête «OU» d'un nœud est une abstraction d'un ensemble de sous-arêtes «ET» tirées simultanément. Ainsi, le choix d'une arête «OU» implique dès lors de se retrouver dans plusieurs états à la fois. Les MDP énumérés sont des exemples de graphes «ET/OU» où chaque arête «OU» est une action correspondant à une liste «ET» de transitions stochastiques vers les états successeurs (cf. figure 7.2).

L'algorithme AO* [93] («AND/OR») a été développé afin de trouver le plus court chemin depuis l'état initial dans un graphe «ET/OU» **acyclique**, c'est-à-dire qui

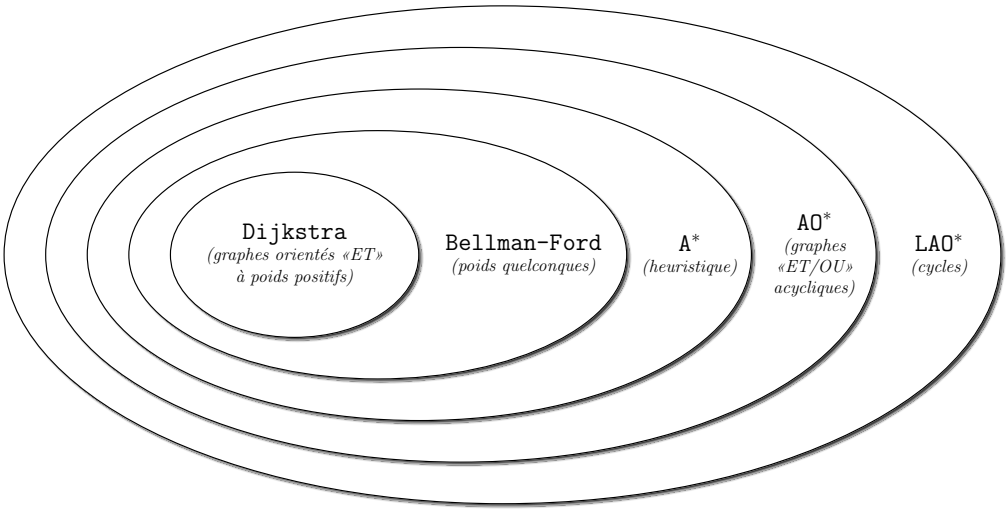


FIG. 7.3 – De Dijkstra à LAO* (propriétés conservées dans les imbrications)

ne contient pas de cycles. Comme A*, AO* utilise une heuristique afin de guider la recherche de la solution. Cependant, les graphes «ET/OU» possèdent souvent des cycles, si bien que l'algorithme LAO* [92] (AO* with Loops) a été récemment proposé afin de traiter les graphes «ET/OU» **cycliques**. De la même manière qu'en programmation dynamique stochastique, la mise à jour de la valeur des nœuds impose des itérations successives (loops) sur tous les nœuds du graphe jusqu'à convergence des valeurs.

Ainsi, lorsque l'état initial est donné, LAO* permet de réduire considérablement la taille de l'espace d'états exploré au cours de la résolution d'un MDP énuméré. Les espaces d'états des MDP factorisés étant souvent très grand en raison de l'explosion combinatoire induite par les variables d'état, LAO* a été naturellement adapté aux MDP factorisés : sLAO* [91] est donc un algorithme de résolution heuristique des MDP factorisés connaissant l'état initial du processus. La figure 7.3 représente l'imbrication des algorithmes suivant les problèmes rencontrés.

7.2 sLAO*

sLAO* est un algorithme itératif (cf. figure 7.4) qui alterne deux phases :
- expansion de l'espace d'états atteignables courant avec la politique courante,
- optimisation du MDP (itération de la valeur) sur le sous-espace d'états atteignables courant, afin de mettre à jour la politique courante.

L'algorithme s'arrête lorsque l'espace d'états atteignables n'augmente plus *et* lorsque l'erreur de Bellman est inférieure à un ϵ donné. Il est **optimal** car la politique obtenue est stable et optimale dans le sous-espace atteignable. Cependant, comme nous le

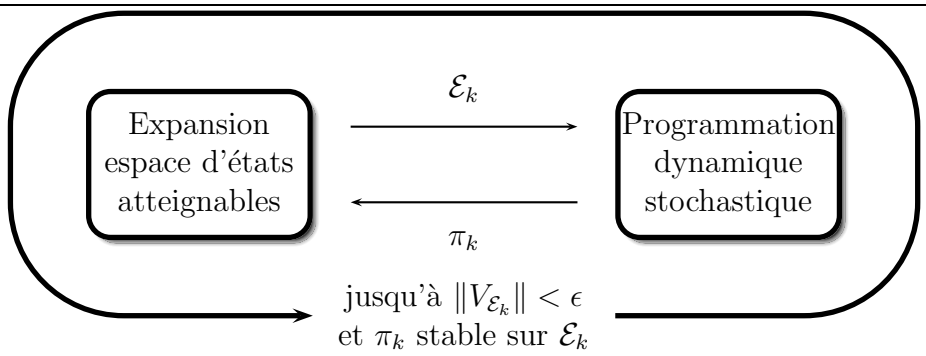


FIG. 7.4 – Schéma itératif de sLAO*

verrons plus loin, sLAO* peut explorer dans le pire des cas tout l'espace d'états, suivant où se situent les récompenses par rapport à l'état initial.

La fonction principale sLAO* est donnée dans l'algorithme 23. Elle prend en entrée le MDP factorisé $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, un ensemble d'états initiaux $\mathcal{I}$ (BDD) ainsi qu'une heuristique h (ADD) qui associe une valeur factorisée à tous les états du MDP. L'algorithme sLAO* itère sur trois ensembles d'états, représentés sous forme de BDD :

- $\mathcal{U}$: frontière de l'ensemble d'états atteignables, c'est-à-dire les *nouveaux* états atteignables,
- $\mathcal{Z}$: ensemble d'états atteignables par toutes les politiques obtenues,
- $\mathcal{W}$: ensemble d'états atteignables par la politique courante restreinte à $\mathcal{Z}$ ($\mathcal{W} \subset \mathcal{Z}$).

Une des conditions d'arrêt de l'algorithme est bien que la politique soit stable :

$$\pi(\mathcal{Z}) \subset \mathcal{Z} \quad \Longleftrightarrow \quad \mathcal{U} = \emptyset$$

La politique est encodée sous forme d'une liste de BDD qui représentent pour chaque action l'ensemble des états où l'action doit être appliquée. Enfin, l'heuristique h est stockée dans la fonction de valeur V (ADD) dès le début de l'algorithme. Ainsi, l'ADD V représente la fonction de valeur exacte à l'intérieur de l'espace d'états atteignables, et l'heuristique h à l'extérieur de cet espace d'états. En particulier, l'heuristique stockée dans une partie de V guide la recherche de la solution dans la frontière $\mathcal{U}$.

7.2.1 Fonction ExpansionEspaceEtatsAtteignables

L'expansion de l'espace d'états atteignables est itérative. Elle repart toujours depuis l'ensemble d'états initiaux $\mathcal{I}$, et à l'intérieur de l'ancien espace d'états attei-

Algorithme 23 : Fonction sLAO*[91]

Données : $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, $\mathcal{I}$, h, γ, ϵ
Résultat : π, V, $\mathcal{W}$, $\mathcal{Z}$
début
  $V \leftarrow h$;
  $\mathcal{Z} \leftarrow$ `ReadLogicZero`();
  pour $a \in \mathcal{A}$ **faire**
    $\pi[a] \leftarrow$ `ReadLogicZero`();
  répéter
    $(\mathcal{U}, \mathcal{W}, \mathcal{Z}) \leftarrow$ `ExpansionEspaceEtatsAtteignables`$(\pi, \mathcal{I}, \mathcal{Z})$;
    $(V, \pi) \leftarrow$ `ProgrammationDynamique`$(\mathcal{W}, V)$;
  jusqu'à $\mathcal{U} =$ `ReadLogicZero`();
  retourner $(\pi, V, \mathcal{W}, \mathcal{Z})$;
fin

gnables $\mathcal{Z}$. Comme le montre la figure 7.5, elle développe itérativement une frontière d'états atteignables partagée en deux sous-espaces :

- $from$: états atteignables à l'intérieur de $\mathcal{Z}$ (anciens états atteignables), à partir desquels sont propagés les états atteignables suivants,
- $\mathcal{U}$: états atteignables à l'extérieur de $\mathcal{Z}$ (nouveaux états atteignables), qui constituent la frontière finale des nouveaux états atteignables $\mathcal{U} \setminus \mathcal{Z}$.

L'expansion des états atteignables s'arrête lorsque $from$ est vide, ce qui peut survenir dans les deux cas de sortie suivants :

- $to \cup \mathcal{W}$ recouvre $\mathcal{Z}$ à l'avant-dernière itération (tous les états qui étaient atteignables le restent), d'où $\mathcal{W} = \mathcal{Z}$ en sortie
- $to \cap \mathcal{Z}$ est vide (des états qui étaient atteignables ne le sont plus), d'où $\mathcal{W} \subsetneq \mathcal{Z}$ en sortie

La fonction `ExpansionEspaceEtatsAtteignables` est détaillée dans l'algorithme 24. L'image d'un ensemble d'états par le BDD dT^a des transitions déterministes d'une action a est calculée par la fonction `ImageAction` présentée plus loin. Aussi, afin d'obtenir l'image de l'ensemble $from$ par une politique π, il suffit de calculer l'union sur toutes les actions de l'image de $from$ restreint aux états où s'applique chaque action de π ($from\_policy$). D'autre part, la différence ensembliste est calculée à l'aide des opérateurs ET puis OU exclusif ($\sqcup$) :

$$to \setminus \mathcal{Z} \quad \Longleftrightarrow \quad to \sqcup (to \cap \mathcal{Z})$$

Du point de vue des BDD, un ensemble d'états E est représenté sous la forme d'une fonction caractéristique χ_E qui vaut 1 sur E et 0 ailleurs. Ainsi, l'image F

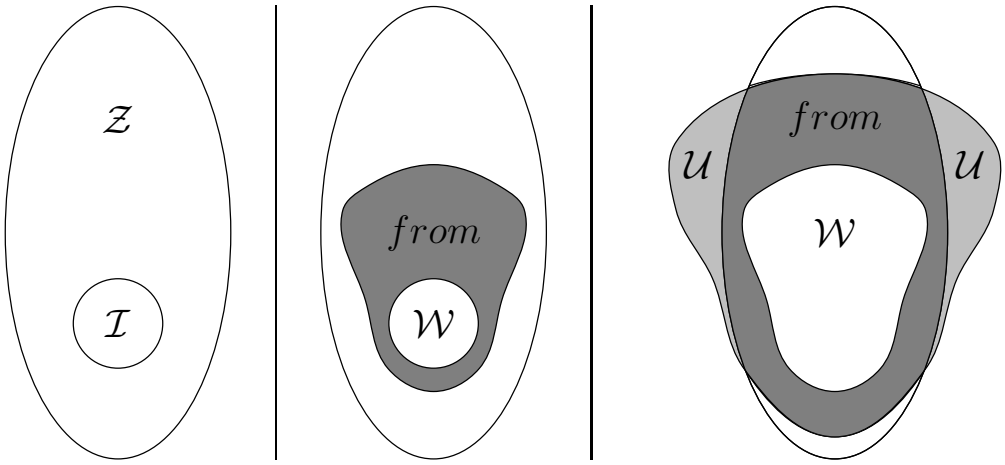


FIG. 7.5 – 2 itérations de `ExpansionEspaceEtatsAtteignables` (sLAO*)

Algorithme 24 : Fonction `ExpansionEspaceEtatsAtteignables` (sLAO*) [91]

Données : π, $\mathcal{I}$, $\mathcal{Z}$
Résultat : $\mathcal{U}$, $\mathcal{W}$, $\mathcal{Z}$
début
  $\mathcal{U} \leftarrow$ `ReadLogicZero`();
  $\mathcal{W} \leftarrow$ `ReadLogicZero`();
  $from \leftarrow \mathcal{I}$;
  répéter
    $to \leftarrow$ `ReadLogicZero`();
    pour $a \in \mathcal{A}$ **faire**
      $from\_policy \leftarrow$ `bddAnd`$(from, \pi[a])$;
      $to \leftarrow$ `bddOr`$(to,$ `ImageAction`$(from\_policy, dT^a))$;
    $to\_and\_\mathcal{Z} \leftarrow$ `bddAnd`$(to, \mathcal{Z})$;
    $\mathcal{U} \leftarrow$ `bddOr`$(\mathcal{U},$ `bddXor`$(to, to\_and\_\mathcal{Z}))$;
    $\mathcal{W} \leftarrow$ `bddOr`$(\mathcal{W}, from)$;
    $from \leftarrow$ `bddXor`$(to\_and\_\mathcal{Z},$ `bddAnd`$(to\_and\_\mathcal{Z}, \mathcal{W}))$;
  jusqu'à $from =$ `ReadLogicZero`();
  $\mathcal{W} \leftarrow$ `bddOr`$(\mathcal{W},\mathcal{U})$;
  $\mathcal{Z} \leftarrow$ `bddOr`$(\mathcal{Z},\mathcal{U})$;
  retourner $(\mathcal{W},\mathcal{U},\mathcal{Z})$;
fin

d'un ensemble E par les transitions déterministes dT^a d'une action a (cf. algorithme 25) est calculée par la formule :

$$\forall\, v' \in \mathcal{V}\ ,\ \chi_F\left(\bigwedge_{i=1}^{n} X'_i = v'_i\right) = \bigcup_{v\in\mathcal{V}}\left(d\mathcal{T}^a\left(\bigwedge_{i=1}^{n} X_i = v_i, \bigwedge_{i=1}^{n} X'_i = v'_i\right) \cap \chi_E\left(\bigwedge_{i=1}^{n} X_i = v_i\right)\right)$$

L'union sur les valeurs non primées est calculée par la fonction `bddExistAbstract`. Les nœuds du BDD obtenu étant des variables primées, une permutation doit être effectuée afin de «déprimer» tous les nœuds.

Algorithme 25 : Fonction `ImageAction`

Données : E (ensemble de départ), dT^a (BDD de transitions déterministes de l'action a)
Résultat : F (ensemble d'arrivée)
début
  $bdd\_temp \leftarrow$ `bddAnd`(E, dT^a);
  $unprimed\_cube \leftarrow$ `bddCube` $(\wedge_{i=1}^{n}\{X_i\})$;
  $bdd\_temp \leftarrow$ `bddExistAbstract`$(bdd\_temp, unprimed\_cube)$;
  $F \leftarrow$ `bddPermute`$(bdd\_temp, primed\_to\_unprimed)$;
  retourner F;
fin

7.2.2 Fonction `ProgrammationDynamique`

La fonction `ProgrammationDynamique` est une version modifiée des algorithmes d'itération de la politique ou de la valeur afin de mettre à jour la fonction de valeur uniquement sur l'espace d'états atteignables courant $\mathcal{W}$. La version proposée dans [91] (cf. algorithme 26) est basée sur SPUDD [32] (itération de la valeur). La programmation dynamique de sLAO*appelle deux fonctions clefs :

- `RestreintTransitions` : restreint les transitions (ADD de récompenses et de probabilités) aux états atteignables $\mathcal{W}$ ainsi qu'à leur frontière $\mathcal{W}'$ (cf. figure 7.6),
- `ActualiseFonctionValeur` : actualise la fonction de valeur entre deux itérations en optimisant sur $\mathcal{W}$ et en appliquant l'heuristique h sur $\mathcal{W}'$.

La nouvelle politique étant en cours de construction, les états atteignables depuis $\mathcal{W}$ doivent être générés pour toutes les actions (espace d'états $\mathcal{W}'$, cf. figure 7.6). Aussi, l'ensemble d'états atteignables $\mathcal{U}$, obtenus durant l'expansion des états en suivant la politique précédente (cf. algorithme 24), ne peut pas être réutilisé dans le calcul de la nouvelle politique : en effet, $\mathcal{U}$ n'a pas été généré en appliquant toutes les actions possibles depuis $\mathcal{I}$.

Algorithme 26 : Fonction `ProgrammationDynamique` (sLAO*) [91]

Données : $\mathcal{W}$ (BDD), V (ADD)
Résultat : V, π (liste de BDD)
début
  $saveV \leftarrow V$;
  $\overline{\mathcal{W}} \leftarrow \texttt{bddXor}(\texttt{ReadOne}(), \mathcal{W})$;
  $saveV_{\overline{\mathcal{W}}} \leftarrow \texttt{addApply}(\texttt{addTimes}, saveV, \texttt{BddToAdd}(\overline{\mathcal{W}}))$;
  $\mathcal{W}' \leftarrow \texttt{ReadLogicZero}()$;
  pour $a \in \mathcal{A}$ **faire**
    $\mathcal{W}' \leftarrow \texttt{bddOr}(\mathcal{W}', \texttt{ImageAction}(\mathcal{W}, d\mathcal{T}^a))$;
  $(R_{\mathcal{W}\cup\mathcal{W}'}, T_{\mathcal{W}\cup\mathcal{W}'}) \leftarrow \texttt{RestreintTransitions}(\mathcal{W}, \mathcal{W}')$;
  $primed\_variables \leftarrow \texttt{addCube}\,(\wedge_{i=1}^{n}\{X_i'\})$;
  répéter
    $V' \leftarrow \texttt{addPermute}(V, unprimed\_to\_primed)$;
    $V \leftarrow$
    $\texttt{ActualiseFonctionValeur}(V', saveV_{\overline{\mathcal{W}}}, \mathcal{W}, R_{\mathcal{W}\cup\mathcal{W}'}, T_{\mathcal{W}\cup\mathcal{W}'}, \textbf{false})$;
    $V_{\mathcal{W}} \leftarrow \texttt{addApply}(\texttt{addTimes}, V, \texttt{BddToAdd}(\mathcal{W}))$;
    $V'_{\mathcal{W}} \leftarrow \texttt{addApply}(\texttt{addTimes}, V', \texttt{BddToAdd}(\mathcal{W}))$;
  jusqu'à $\texttt{EqualSupNorm}(V_{\mathcal{W}}, V'_{\mathcal{W}}, \epsilon) = \textbf{true}$;
  $(V, \pi) \leftarrow$
  $\texttt{ActualiseFonctionValeur}(V', saveV_{\overline{\mathcal{W}}}, \mathcal{W}, R_{\mathcal{W}\cup\mathcal{W}'}, T_{\mathcal{W}\cup\mathcal{W}'}, \textbf{true})$;
  retourner (V, π);
fin

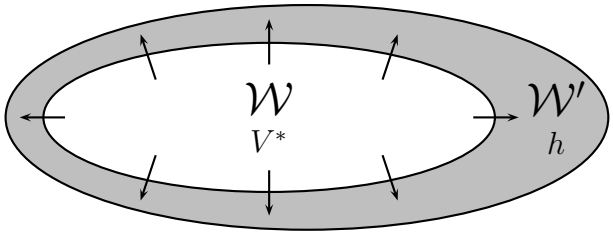


FIG. 7.6 – Programmation dynamique dans sLAO* : l'heuristique est utilisée sur la frontière de l'espace d'états atteignables courant

Fonction `RestreintTransitions`

La fonction `RestreintTransitions` (cf. algorithme 27) restreint les ADD de transitions R^a et T^a d'une action a à l'ensemble des états atteignables et de leur frontière : $\mathcal{W} \cup \mathcal{W}'$. Comme ces ADD contiennent des variables non primées (états de départ) et des variables primées (états d'arrivée), la restriction est obtenue en multipliant successivement chaque ADD par la fonction caractéristique $\chi_{\mathcal{W}\cup\mathcal{W}'}$ et sa version «primée» $\chi'_{\mathcal{W}\cup\mathcal{W}'}$.

Algorithme 27 : Fonction `RestreintTransitions`

Données : R (ADD), T (ADD), $\mathcal{W}$ (BDD), $\mathcal{W}'$ (BDD)
Résultat : $R_{\mathcal{W}\cup\mathcal{W}'}$ (ADD), $T_{\mathcal{W}\cup\mathcal{W}'}$ (ADD)
début
  $add\_union \leftarrow$ `BddToAdd`(`bddOr`$(\mathcal{W}, \mathcal{W}')$);
  $add\_union\_bis \leftarrow$ `addPermute`$(add\_union, unprimed\_to\_primed)$;
  pour $a \in \mathcal{A}$ **faire**
    $R^a_{\mathcal{W}\cup\mathcal{W}'} \leftarrow$ `addApply`(`addTimes`$, R^a, add\_union$);
    $R^a_{\mathcal{W}\cup\mathcal{W}'} \leftarrow$ `addApply`(`addTimes`$, R^a_{\mathcal{W}\cup\mathcal{W}'}, add\_union\_bis$);
    $T^a_{\mathcal{W}\cup\mathcal{W}'} \leftarrow$ `addApply`(`addTimes`$, T^a, add\_union$);
    $T^a_{\mathcal{W}\cup\mathcal{W}'} \leftarrow$ `addApply`(`addTimes`$, T^a_{\mathcal{W}\cup\mathcal{W}'}, add\_union\_bis$);
  retourner $(R_{\mathcal{W}\cup\mathcal{W}'}, T_{\mathcal{W}\cup\mathcal{W}'})$;
fin

Fonction `ActualiseFonctionValeur`

Les états atteignables de $\mathcal{W}$ sont évalués en considérant que la valeur des états successeurs *toutes actions confondues* (sous-espace d'états $\mathcal{W}'$) est donnée par l'heuristique h (cf. figure 7.6). La multiplication de la fonction de valeur maximum (add_max) par la fonction caractéristique $\chi_{\mathcal{W}}$ (cf. algorithme 28) permet de limiter la mise à jour de la fonction de valeur sur $\mathcal{W}$. De même, $saveV_{\overline{\mathcal{W}}}$ a été obtenu en multipliant l'ancienne fonction de valeur par $\chi_{\overline{\mathcal{W}}}$ (cf. algorithme 26). Ainsi, la valeur des états $saveV_{\overline{\mathcal{W}}}$ qui ne sont pas dans $\mathcal{W}$, et en particulier ceux de $\mathcal{W}'$, est bien l'heuristique h qui a initialisé la fonction de valeur V dans l'algorithme principal 23.

Pour des raisons d'efficacité, la politique n'est calculée qu'à la demande ($calcule\_politique$ = **true**). Nous précisons que le calcul de la politique n'est pas détaillé dans [91], si bien que la version proposée dans l'algorithme 28 n'est pas nécessairement celle utilisée dans [91].

Pour une action a, la différence ($diff$) entre le maximum de la fonction de valeur V^a et le maximum courant (add_max) est un ADD dont les valeurs nulles

Algorithme 28 : Fonction `ActualiseFonctionValeur` (sLAO*)

Données : V' (ADD avec variables primées), $saveV_{\overline{\mathcal{W}}}$ (ADD), $\mathcal{W}$ (BDD), $R_{\mathcal{W}\cup\mathcal{W}'}$ (ADD), $T_{\mathcal{W}\cup\mathcal{W}'}$ (ADD), *calcule_politique* (booléen)
Résultat : V (ADD avec variables non primées), $[\pi]$ (liste de BDD, retourné si *calcule_politique* = **true**)

début
  pour $a \in \mathcal{A}$ **faire**
    $V^a \leftarrow \texttt{addApply}(\texttt{addTimes}, T^a_{\mathcal{W}\cup\mathcal{W}'}, V')$;
    $V^a \leftarrow \texttt{addExistAbstract}(V^a, primed\_variables)$;
    $V^a \leftarrow \texttt{addApply}(\texttt{addTimes}, \texttt{addConst}(\gamma), V^a)$;
    $V^a \leftarrow \texttt{addApply}(\texttt{addPlus}, V^a, R^a_{\mathcal{W}\cup\mathcal{W}'})$;
    si $a = \mathcal{A}.debut()$ **alors**
      $add\_max \leftarrow V^a$;
      si *calcule_politique* = **true alors**
        $\pi[a] \leftarrow \mathcal{W}$;
    sinon
      $add\_max \leftarrow \texttt{addApply}(\texttt{addMaximum}, add\_max, V^a)$;
      si *calcule_politique* = **true alors**
        $diff \leftarrow \texttt{addApply}(\texttt{addMinus}, add\_max, V^a)$;
        $\pi[a] \leftarrow \texttt{bddXor}(\mathcal{W}, \texttt{bddAnd}(\texttt{addBddPattern}(diff), \mathcal{W}))$;
        pour $a' = \mathcal{A}.debut()$ **jusqu'à** $a' = \mathcal{A}.precedent(a)$ **faire**
          $\pi[a'] \leftarrow \texttt{bddXor}(\pi[a'], \texttt{bddAnd}(\pi[a'], \pi[a]))$;
  $add\_max\_\mathcal{W} \leftarrow \texttt{addApply}(\texttt{addTimes}, add\_max, \texttt{BddToAdd}(\mathcal{W}))$;
  $V \leftarrow \texttt{addApply}(\texttt{addPlus}, add\_max\_\mathcal{W}, saveV_{\overline{\mathcal{W}}})$;
  si *calcule_politique* = **true alors**
    retourner (V, π);
  sinon
    retourner V;
fin

correspondent aux états où l'action a maximise la fonction de valeur. Ainsi, en mettant à 1 toutes les valeurs non nulles de cet ADD (fonction `addBddPattern`), le complémentaire sur $\mathcal{W}$ du BDD obtenu ($\pi[a]$) vaut 1 sur les états où l'action a maximise la fonction de valeur. Enfin, pour que les actions de la politique π soient exclusives, il convient de soustraire $\pi[a]$ des BDD des actions précédemment traitées ($\pi[a']$).

7.2.3 Heuristique

L'heuristique admissible utilisée dans [91] est une fonction de valeur approximative, calculée préalablement en utilisant des algorithmes d'optimisation approximative de MDP tels APRICODD [33] ou [28, 89]. L'heuristique sert à évaluer les *nouveaux* états de l'espace d'états atteignables courant, qui constituent la *frontière* de l'espace d'états atteignables (cf. figure 7.6).

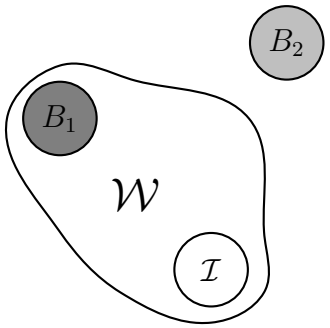
Bien entendu, d'autres heuristiques sont possibles en fonction du problème rencontré. L'heuristique précédente est peu adaptée à des sous-buts imposés car elle focalise la recherche vers les zones de plus forte récompenses qui ne sont pas nécessairement les sous-buts à atteindre. Aussi, la politique trouvée par sLAO* risque de ne pas satisfaire la contrainte de réalisation des sous-buts. Ainsi, dans le chapitre suivant, nous proposerons une heuristique adaptée à l'atteignabilité de sous-buts imposés par la mission.

7.3 De sLAO* à sfDP

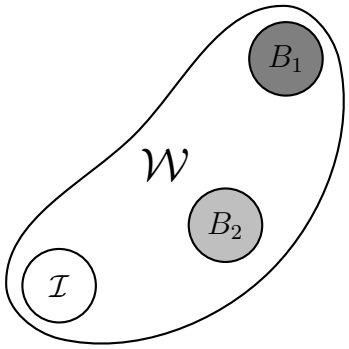
7.3.1 sLAO* limite les états explorés

Lorsqu'un ensemble d'états initiaux est donné, sLAO* restreint l'espace d'états explorés aux états atteignables depuis les états initiaux. Dans la phase d'expansion de l'espace d'états atteignable, les états atteignables sont propagés depuis les états initiaux jusqu'à recouvrir le précédent ensemble d'états atteignables ou jusqu'à sortir de cet ensemble. Ensuite, dans la phase de programmation dynamique, la politique partielle est mise à jour en étant guidée par l'heuristique sur la frontière d'états atteignables.

Il a été démontré que sLAO* est nettement plus performant que les algorithmes optimaux comme SPI [27] ou SPUDD [32] dans la plupart des cas. Une généralisation en ligne appelée sRTDP, qui partage des notions communes avec sLAO*, a été développée [95]. Comme sLAO*, sRTDP est une version symbolique d'un algorithme de programmation dynamique en ligne sur espace d'états énumérés, décliné en deux classes : RTDP [96] et LRTDP [97, 98]. La version symbolique s'est avérée également supérieure en performances à la version énumérée.



B_1 et B_2 sont tous les deux imposés : π évite certains sous-buts



seul B_2 est imposé : π développe trop d'états atteignables

FIG. 7.7 – Deux cas défavorables de sLAO* ($r(B_1) \gg r(B_2)$)

7.3.2 sLAO* ne cible pas la recherche des états explorés

L'heuristique est une approximation factorisée de la fonction de valeur sur tout l'espace d'états. Par conséquent, la politique construite est guidée vers les zones de plus fortes récompenses, indépendamment des sous-buts de planification imposés par la mission. Ainsi, comme le montre la figure 7.7, deux cas défavorables peuvent survenir :

- soit la politique partielle évite certains sous-buts et l'espace d'états atteignables ne contient pas tous les sous-buts,
- soit la politique partielle «passe» par les sous-buts mais en continuant jusqu'aux zones de plus fortes récompenses, si bien que l'espace d'états atteignables développé contient trop d'états.

D'autre part, l'heuristique est calculée avec des algorithmes approximatifs de programmation dynamique stochastique tels APRICODD [33]. Aussi, sLAO* est limité par le calcul de l'heuristique, optimisée sur l'espace d'états entier. Si, à cause de l'explosion combinatoire due aux variables d'état, l'heuristique ne peut pas être calculée, sLAO* ne pourra pas être lancé.

7.3.3 sfDP : un algorithme qui cible les sous-buts imposés par la mission

Afin de remédier aux deux problèmes précédents, nous proposons dans les chapitres suivants un nouvel algorithme de programmation dynamique stochastique symbolique, heuristique et focalisé, appelé sfDP («Symbolic Focused Dynamic Programming»). Comme sLAO*, notre algorithme calcule d'abord une heuristique admissible, puis alterne une phase de programmation dynamique sur un sous-espace

d'états atteignables et une phase d'expansion de cet espace d'états atteignables avec la politique courante.

Cependant, contrairement à sLAO*, sfDP cible la recherche des états atteignables vers les sous-buts imposés par la mission, qui ne sont pas nécessairement les zones de plus fortes récompenses du problème. Les deux principales différences avec sLAO* sont :

- l'heuristique que nous proposons favorise les états qui ont une grande probabilité d'atteindre les sous-buts, et non ceux qui ont une grande valeur au regard des zones de plus fortes récompenses,
- l'expansion de l'espace d'états atteignables avec la politique courante s'arrête lorsque les sous-buts ont été atteints, et non lorsque le précédent espace d'états atteignables a été recouvert.

Chapitre 8

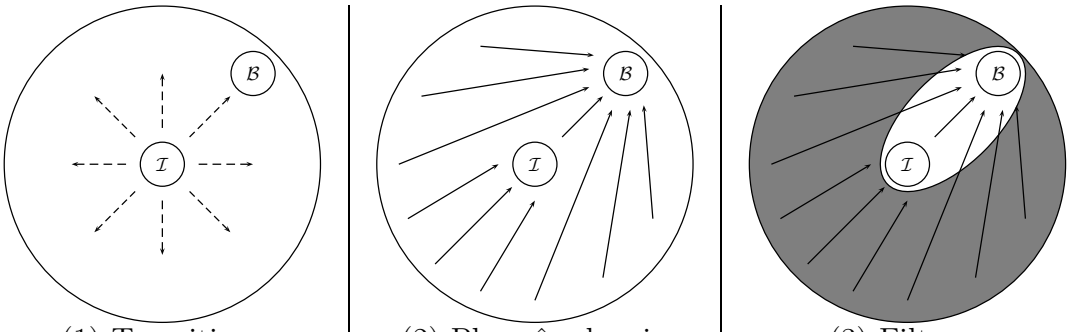
Recherche heuristique des sous-buts de planification

8.1 Une heuristique admissible qui focalise l'espace de recherche sur les sous-buts de la mission

Dans ce chapitre, nous proposons une heuristique qui cible les sous-buts imposés par la mission, afin que l'espace d'états atteignables soit focalisé sur les sous-buts imposés par la mission. Contrairement à l'heuristique présentée dans `sLAO*` [91, 92], qui est approximation de la fonction de valeur définie sur tout l'espace d'états, notre heuristique est une politique de «plus sûr chemin stochastique» définie à l'intérieur d'un sous-ensemble d'états permettant d'atteindre les sous-buts avec une probabilité suffisante. Le calcul de l'heuristique est divisé en trois parties successives (cf. figure 8.1) :

1. Calcul d'un sous-espace d'états atteignables initial contenant les états initiaux et les sous-buts, obtenu en rendant les transitions déterministes,
2. Calcul d'une politique de plus sûr chemin stochastique, c'est-à-dire qui maximise la probabilité d'atteindre les sous-buts depuis tous les états du sous-espace d'états atteignables initial,
3. Réduction du sous-espace d'états atteignables par filtrage des états qui ont une faible probabilité d'atteindre les sous-buts comparés aux états initiaux.

Notre heuristique est admissible dans le sens où la politique de plus sûr chemin stochastique garantit de trouver *en moyenne* un chemin sûr vers les sous-buts de la mission. En outre, elle est plus informative que celle de `sLAO*`, car elle fournit à la fois un espace d'états atteignables initial réduit ainsi qu'une politique de plus sûr chemin



(1) Transitions déterministes | (2) Plus sûr chemin stochastique | (3) Filtrage stochastique

FIG. 8.1 – Une heuristique qui cible les sous-buts imposés par la mission en trois étapes

stochastique définie à l'intérieur de ce sous-espace d'états. En revanche, contrairement à l'heuristique de sLAO*, la valeur de notre heuristique est incompatible avec la fonction de valeur du MDP : en effet, à cause des récompenses ou des pénalités du domaine, il n'y a aucune relation évidente a priori entre la valeur optimale d'un état et sa probabilité d'atteindre les sous-buts (d'autant plus que les récompenses ne sont pas nécessairement aux sous-buts). Par conséquent, comme nous le verrons dans le chapitre suivant, notre heuristique ne sera pas utilisée pour évaluer les états sur le frontière de l'espace d'états atteignables courant.

Avant de présenter notre heuristique plus en détail, précisons que nous nous sommes inspirés de travaux sur l'analyse d'atteignabilité déterministe pour les MDP factorisés basés sur les arbres de décision [30]. Dans cet article, un espace d'états atteignables est étendu itérativement depuis un état initial jusqu'à contenir des sous-buts de planification, en utilisant des notions de planification classique (notamment GRAPHPLAN [10] et STRIPS [67]). Néanmoins, contrairement au calcul de notre heuristique, les données sont représentées sous forme d'arbres de décision (vs. BDD ou ADD), et l'analyse d'atteignabilité est uniquement déterministe (vs. probabiliste).

8.2 Expansion de l'espace d'états atteignables par relaxation de contraintes

8.2.1 Relaxation des contraintes

La première phase de notre heuristique consiste à obtenir un ensemble d'états atteignables initial, qui permet d'atteindre les sous-buts $\mathcal{B}$ imposés par la mission, *pour toutes politiques* partant d'états initiaux $\mathcal{I}$.

Les opérations sur les BDD étant plus performantes que celles sur les ADD, nous transformons en BDD tous les ADD représentant les probabilités de transition de chaque action (comme dans sLAO*). Cette *relaxation de contraintes* est effectuée une seule fois à l'aide de la fonction `addBddPattern`, qui met à 1 tous les éléments non nuls d'un ADD. Ainsi, les BDD obtenus $(dT^a)_{a\in\mathcal{A}}$ représentent les matrices d'adjacence creuses des transitions de chaque action.

8.2.2 `ExpansionEspaceEtatsAtteignables` : une fonction à double emploi

Comme dans sLAO*, l'algorithme d'expansion de l'espace d'états atteignables sera réutilisé lors de la résolution du MDP, en alternance avec une phase de programmation dynamique partielle. Néanmoins, l'expansion des états atteignables dans le calcul initial de notre heuristique est effectuée en appliquant toutes les actions possibles (aucune politique n'est encore calculée), alors qu'elle sera réalisée en suivant la politique partielle courante dans la phase de résolution (cf. chapitre suivant). Ainsi, les arguments π et $\mathcal{Z}$ de la fonction `ExpansionEspaceEtatsAtteignables` (cf. algorithme 29) sont optionnels : ils ne sont lus que si la variable booléenne *recalcule_etats_atteignables* est vraie. L'expansion de l'espace d'états atteignables est donc utilisée de deux manières différentes :

- *recalcule_etats_atteignables* = **false** : calcule l'espace d'états atteignables en appliquant toutes les actions (politiques) possibles (uniquement dans le calcul d'heuristique)
- *recalcule_etats_atteignables* = **true** : recalcule l'espace d'états atteignables en suivant la politique partielle courante (uniquement dans la résolution du MDP)

Dans les deux cas, l'expansion de l'espace d'états est réalisée grâce à la fonction `ImageAction` (cf. algorithme 25 page 187) qui calcule l'image d'un ensemble E par la matrice d'adjacence M^a d'une action a. Pour le calcul d'heuristique, M^a est la matrice dT^a des transitions déterministes de l'action a. Lors de la résolution du MDP, M^a est réduite aux transitions déterministes de l'action a pour la politique π : $M^a = dT^a \cap \pi[a]$. En effet, le BDD $\pi[a]$ représente l'ensemble des états où s'applique l'action a dans la politique π.

8.2.3 Une expansion conditionnée par les sous-buts de la mission

Dans sLAO*(cf. algorithme 24 page 186), l'expansion de l'espace d'états s'arrête lorsque tous les états de l'ancien espace d'états atteignables $\mathcal{Z}$ ont été explorés

Algorithme 29 : Fonction `ExpansionEspaceEtatsAtteignables` (`sfDP`)

Données : $\mathcal{I}$ (BDD), $\mathcal{B}$ (BDD), *condition_arret* (booléen : **faible** ou **forte**), *recalcule_etats_atteignables* (booléen), $[\pi]$ (liste optionnelle de BDD), $[\mathcal{Z}]$ (BDD optionnel)
Résultat : $\mathcal{W}$ (BDD), $[\mathcal{U}]$ (BDD optionnel), $[\mathcal{Z}]$ (BDD optionnel)
début
- $\mathcal{W} \leftarrow$ `ReadLogicZero`();
- **si** *recalcule_etats_atteignables* = **true alors**
  - $\mathcal{U} \leftarrow$ `ReadLogicZero`();
- $from \leftarrow \mathcal{I}$;
- *continuer* $\leftarrow$ **true**;
- **tant que** *continuer* = **true faire**
  - $to \leftarrow$ `ReadLogicZero`();
  - **si** *recalcule_etats_atteignables* = **true alors**
    - **pour** $a \in \mathcal{A}$ **faire**
      - $from\_policy \leftarrow$ `bddAnd`$(from, \pi[a])$;
      - $to \leftarrow$ `bddOr`$(to,$ `ImageAction`$(from\_policy, dT^a))$;
  - **sinon**
    - **pour** $a \in \mathcal{A}$ **faire**
      - $to \leftarrow$ `bddOr`$(to,$ `ImageAction`$(from, dT^a))$;
  - $\mathcal{W} \leftarrow$ `bddOr`$(\mathcal{W}, from)$;
  - **si** *recalcule_etats_atteignables* = **true alors**
    - $\mathcal{U} \leftarrow$ `bddOr`$(\mathcal{U},$ `bddXor`$(to,$ `bddAnd`$(to, \mathcal{Z})))$;
  - $from \leftarrow$ `bddXor`$(to,$ `bddAnd`$(to, \mathcal{W}))$;
  - **si** *condition_arret* = **faible alors**
    - **si** `bddAnd`$(to, \mathcal{B}) \neq$ `ReadLogicZero`() **alors**
      - *continuer* = **false**;
  - **sinon**
    - **si** `bddAnd`$(\mathcal{W}, \mathcal{B}) = \mathcal{B}$ **alors**
      - *continuer* = **false**;
- $\mathcal{W} \leftarrow$ `bddOr`$(\mathcal{W}, from)$;
- **si** *recalcule_etats_atteignables* = **true alors**
  - $\mathcal{Z} \leftarrow$ `bddOr`$(\mathcal{Z},\mathcal{U})$;
  - **retourner** $(\mathcal{W},\mathcal{U},\mathcal{Z})$;
- **sinon**
  - **retourner** $\mathcal{W}$;

fin

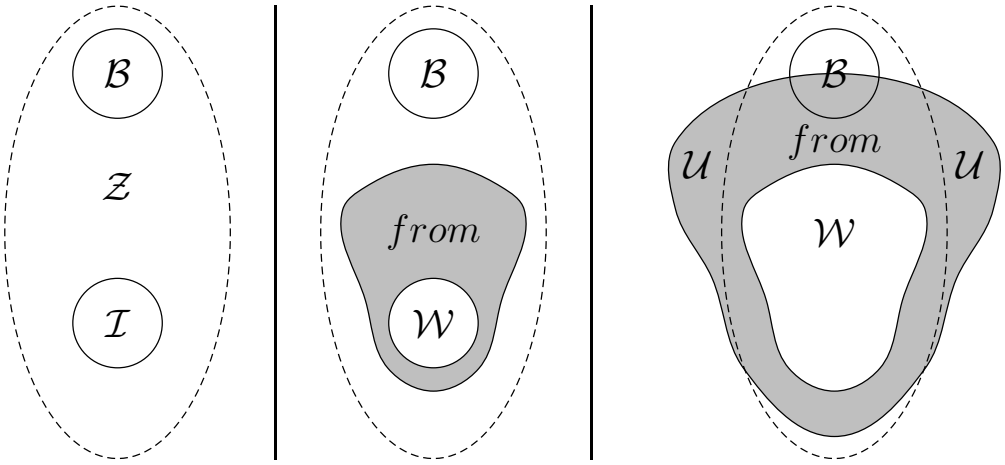


FIG. 8.2 – Deux itérations de `ExpansionEspaceEtatsAtteignables` (`sfDP`), la deuxième représentant une condition d'arrêt **faible** de l'expansion du sous-espace atteignable

($from = \emptyset$). Ainsi, dans `sLAO*`, l'expansion des atteignables est conditionnée par l'ancien espace d'états atteignables.

Au contraire, dans `sfDP`, nous proposons de conditionner l'expansion des états atteignables par les sous-buts de la mission, en arrêtant l'expansion lorsque l'espace d'états atteignables a atteint l'ensemble de sous-buts $\mathcal{B}$. Nous utilisons les mêmes notations que `sLAO*`, mais les ensembles d'état qu'elles désignent dans `sfDP` sont différents :

- $\mathcal{Z}$: ensemble d'états atteignables précédent (utilisé uniquement dans la résolution du MDP),
- $\mathcal{W}$: nouvel ensemble d'états atteignables,
- $\mathcal{U}$: ensemble des nouveaux états atteignables après expansion, c'est-à-dire qui ne sont pas dans $\mathcal{Z}$ (utilisé uniquement dans la résolution du MDP),
- $from$: frontière des états atteignables durant l'expansion,
- to : nouveaux états atteignables durant l'expansion

Contrairement à `sLAO*`, les ensembles $\mathcal{W}$ et $from$ ne sont plus restreints à $\mathcal{Z}$ car l'expansion des états atteignables, dans `sfDP`, n'est pas conditionnée par l'ensemble précédent des états atteignables. En particulier, comme le montre la figure 8.2, $\mathcal{U}$ est inclus dans $from$ mais pas dans $\mathcal{Z}$. La comparaison avec la figure 7.5 correspondante de `sLAO*` (page 186) permet de comprendre les différences entre les deux versions d'expansion de l'espace d'états atteignables.

En revanche, $\mathcal{Z}$ et $\mathcal{U}$ ont la même signification que dans `sLAO*`, car ils permettent de savoir, durant la résolution du MDP, si l'espace d'états atteignables a changé après la précédente phase de programmation dynamique partielle.

8.2.4 Deux conditions d'arrêt différentes

L'ensemble de sous-buts $\mathcal{B}$, comme l'ensemble d'états initiaux $\mathcal{I}$, est une disjonction de conjonctions de variables d'état. La disjonction propose un choix de buts à réaliser, et la conjonction impose que toutes les valeurs des variables d'état d'un choix soient satisfaites.

Les variables d'état de chaque choix constituent un sous-ensemble des variables d'état, si bien que leur conjonction définit un sous-ensemble de l'ensemble d'états. Par conséquent, n'importe quel élément de $\mathcal{B}$ satisfait la réalisation d'un cas de sous-buts imposés par la mission.

Aussi, nous distinguons deux cas de condition d'arrêt (variable booléenne *condition_arret*), que nous comparerons dans le chapitre suivant :

- *condition_arret* = **faible** : l'expansion s'arrête lorsqu'il existe **au moins un** état atteignable qui satisfait tous les sous-buts d'un cas possible de réalisation des sous-buts ($from \cap \mathcal{B} \neq \emptyset$)
- *condition_arret* = **forte** : l'expansion s'arrête lorsque **tous** les états satisfaisant tous les cas possibles de réalisation des sous-buts sont atteignables ($\mathcal{W} \cap \mathcal{B} = \mathcal{B} \Leftrightarrow \mathcal{B} \subset \mathcal{W}$)

Le schéma de droite de la figure 8.2 présente une condition d'arrêt faible.

La condition d'arrêt faible est sous-optimale au regard de la condition d'arrêt forte, car elle envisage beaucoup moins de cas de réalisation des sous-buts. En revanche, elle permet de développer un espace d'états atteignables nettement plus petit.

8.3 Heuristique de plus sûr chemin stochastique

La deuxième étape du calcul de notre heuristique en est le noyau central : c'est elle qui focalise la recherche de la solution vers les sous-buts.

8.3.1 Formalisme

L'espace d'états obtenu précédemment, dans sa version «toutes actions possibles», garantit seulement que toutes les politiques restreintes à cet espace mènent aux sous-buts depuis les états initiaux. L'expansion n'est pas guidée vers les sous-buts car elle est appliquée pour toutes les actions possibles. Ainsi, nous proposons de calculer une politique de «plus sûr chemin stochastique» qui maximise la probabilité d'arriver aux sous-buts en partant de n'importe quel état de l'espace atteignable (cf. définition 13).

Probabilité d'atteindre les sous-buts pour une politique donnée

Pour une politique π donnée, il est possible d'atteindre $\mathcal{B}$ en partant de $\mathcal{I}$. Lorsque l'agent arrive en $\mathcal{B}$, la mission est terminée : l'agent ne passe donc qu'une seule fois dans $\mathcal{B}$ (à l'arrivée). Ainsi, tout chemin partant de $\mathcal{I}$ et arrivant dans $\mathcal{B}$ est dans $\mathcal{W} \setminus \mathcal{B}$. Le théorème 11 qui, non proposé dans la littérature à notre connaissance, prouve alors que la probabilité d'atteindre $\mathcal{B}$ en suivant la politique π, qui est la somme sur tous les chemins possibles de $\mathcal{W} \setminus \mathcal{B}$ menant à $\mathcal{B}$, converge.

La démonstration de notre théorème repose sur la restriction de la matrice P_π des transitions stochastiques de la politique π au sous-ensemble $\mathcal{S}$ des états de $\mathcal{W} \setminus \mathcal{B}$ pour lesquels il existe un chemin de probabilité non nulle menant à $\mathcal{B}$. Aucun état de $\mathcal{S}$ n'est absorbant (sinon la probabilité d'arriver dans $\mathcal{B}$ depuis cet état serait nulle), si bien que tous les coefficients de la matrice $P_{\pi|_\mathcal{S}}$ sont strictement inférieurs à 1 : cette matrice est donc contractante (norme strictement inférieure à 1), si bien qu'«elle joue le rôle» du coefficient d'actualisation $\gamma < 1$ dans la contraction de l'opérateur de Bellman [14].

A titre de comparaison, la probabilité agrégée d'obtenir une valeur x_i d'une variable X_i — pour la réduction automatique de l'arité d'une variable d'état X_p (cf. théorème 10) — ne converge pas nécessairement car le sous-espace d'états $X_i = x_i$ n'est pas absorbant. Il est en effet possible que la valeur de X_i change par la suite au cours de la réalisation de la politique locale, contrairement au sous-espace d'états $\mathcal{B}$ dans le cas présent.

Théorème 11 *Probabilité d'atteindre les sous-buts*

Soit $\mathcal{B}$ un sous-ensemble d'états buts de $\mathcal{V}$.
Soit $\mathcal{W}$ un sous-ensemble d'états atteignables de $\mathcal{V}$.
Soit $\mathcal{S}$ le sous-ensemble des états de $\mathcal{W}$ pour lesquels il existe au moins un chemin de probabilité non nulle menant à $\mathcal{B}$.
Soit π une politique définie sur $\mathcal{W}$, de matrice de transitions P_π.
Soit χ_E la fonction indicatrice d'un sous-ensemble d'états E.
La probabilité P_w d'atteindre $\mathcal{B}$ depuis n'importe quel état w de $\mathcal{W}$ vaut :

$$P_w = \chi_{\mathcal{B}\cap\mathcal{W}}(w) + \chi_\mathcal{S}(w) \cdot \left(\sum_{(w',w'')\in\mathcal{S}\times\mathcal{B}} \left(\mathbf{I}_\mathcal{S} - P_{\pi|_\mathcal{S}}\right)^{-1}(w,w') \cdot P_\pi(w',w'') \right)$$

Démonstration. Soit $w \in \mathcal{W}$.
Si $w \in \mathcal{B}$, les sous-buts sont atteints dès le départ donc $P_w = 1$ sur $\mathcal{B} \cap \mathcal{W}$.
Si $w \notin \mathcal{B}$, les sous-buts sont atteints en plus d'un coup. La probabilité de les atteindre est donc la somme, sur tous les chemins de longueur supérieure à 1 dont seul le dernier

état est dans $\mathcal{B}$, d'atteindre $\mathcal{B}$ sur chaque chemin.
Calculons la probabilité d'atteindre $\mathcal{B}$ une seule fois en t coups, $t > 0$:

$$\begin{aligned}
p_w(t) &= P\left(w_t \in \mathcal{B} \mid \pi, w_0 = w, (w_1, \ldots, w_{t-1}) \in (\mathcal{W} \setminus \mathcal{B})^{t-1}\right) \\
&= \sum_{w'' \in \mathcal{B}} P\left(w_t = w'' \mid \pi, w_0 = w, (w_1, \ldots, w_{t-1}) \in (\mathcal{W} \setminus \mathcal{B})^{t-1}\right) \\
&= \sum_{w'' \in \mathcal{B}} \sum_{w' \in \mathcal{W} \setminus \mathcal{B}} P\left(w_t = w'' \mid \pi, w_{t-1} = w'\right) \cdot \\
&\qquad\qquad P\left(w_{t-1} = w' \mid \pi, w_0 = w, (w_1, \ldots, w_{t-2}) \in (\mathcal{W} \setminus \mathcal{B})^{t-2}\right)
\end{aligned}$$

Or, $P\left(w_{t-1} = w' \mid \pi, w_0 = w, (w_1, \ldots, w_{t-2}) \in (\mathcal{W} \setminus \mathcal{B})^{t-2}\right)$, avec $(w, w_{t-1}) \in (\mathcal{W} \setminus \mathcal{B})^2$, est la probabilité de passer de w à w_{t-1} en $t-1$ coups dans la sous-chaîne de Markov définie sur $\mathcal{W} \setminus \mathcal{B}$. Cette probabilité est donc $\left(P_{\pi|_{\mathcal{W}\setminus\mathcal{B}}}\right)^{t-1}$, où $P_{\pi|_{\mathcal{W}\setminus\mathcal{B}}}$ est la matrice de transitions de P_π restreinte à $\mathcal{W} \setminus \mathcal{B}$. Ainsi :

$$p_w(t) = \sum_{(w', w'') \in (\mathcal{W}\setminus\mathcal{B}) \times \mathcal{B}} \left(P_{\pi|_{\mathcal{W}\setminus\mathcal{B}}}\right)^{t-1} (w, w') P_\pi(w', w'')$$

Soit $T > 0$, montrons que la série $\sum_{t=1}^{T} p_w(t)$ converge, en séparant les états de $\mathcal{W} \setminus \mathcal{B}$ qui mènent à $\mathcal{B}$ en un nombre fini de coups (sous-ensemble d'état $\mathcal{S}$), des autres états de $\mathcal{W} \setminus \mathcal{B}$. Commençons par démontrer par récurrence sur t que :

$$\forall\, t > 0\ ,\ p_w(t) = \chi_{\mathcal{S}}(w) \cdot \sum_{(w', w'') \in \mathcal{S} \times \mathcal{B}} \left(P_{\pi|_{\mathcal{S}}}\right)^{t-1} (w, w') \cdot P\left(w'' \mid \pi(w'), w'\right)$$

– rang 1 :
Si $w \in \mathcal{S}$:

$$\begin{aligned}
p_w(1) &= \sum_{w'' \in \mathcal{B}} P\left(w'' \mid \pi(w), w\right) \\
&= \sum_{w' \in \mathcal{W}\setminus\mathcal{B}} \sum_{w'' \in \mathcal{B}} \delta_w(w') \cdot P\left(w'' \mid \pi(w), w\right) \\
&= \sum_{(w', w'') \in \mathcal{W}\setminus\mathcal{B} \times \mathcal{B}} \mathbf{I}_{\mathcal{W}\setminus\mathcal{B}}(w, w') \cdot P\left(w'' \mid \pi(w), w\right) \\
&= \sum_{(w', w'') \in \mathcal{S} \times \mathcal{B}} \mathbf{I}_{\mathcal{S}}(w, w') \cdot P\left(w'' \mid \pi(w), w\right) \qquad \text{car } w \in \mathcal{S} \\
&= \sum_{(w', w'') \in \mathcal{S} \times \mathcal{B}} \left(P_{\pi|_{\mathcal{S}}}\right)^0 (w, w') \cdot P\left(w'' \mid \pi(w), w\right)
\end{aligned}$$

Si $w \notin \mathcal{S} : \forall\, w' \in \mathcal{B}\ ,\ P(w' \mid \pi(w), w) = 0$, donc :

$$p_w(1) = \sum_{w' \in \mathcal{B}} P(w' \mid \pi(w), w) = 0$$

– supposons la propriété de récurrence vraie au rang t :

$$\begin{aligned} p_w(t+1) &= \sum_{(w',w'') \in (\mathcal{W}\setminus\mathcal{B})\times\mathcal{B}} \left(P_{\pi|_{\mathcal{W}\setminus\mathcal{B}}}\right)^t (w,w') P_\pi(w',w'') \\ &= \sum_{(w',w'',w''') \in (\mathcal{W}\setminus\mathcal{B})^2\times\mathcal{B}} P\left(w''' \mid \pi(w), w\right) \left(P_{\pi|_{\mathcal{W}\setminus\mathcal{B}}}\right)^{t-1} (w''',w') P_\pi(w',w'') \\ &= \sum_{w''' \in \mathcal{W}\setminus\mathcal{B}} P\left(w''' \mid \pi(w), w\right) p_{w'''}(t) \end{aligned}$$

Si $w \notin \mathcal{S}$:
– Si $w''' \in \mathcal{S}$ $(p_{w'''}(t) > 0)$ alors $P(w''' \mid \pi(w), w) = 0$ car, sinon, il y aurait un chemin de longueur finie menant de w à $\mathcal{B}$
– Si $w''' \notin \mathcal{S}$ alors $p_{w'''}(t) = 0$
Donc $p_w(t+1) = 0$
Si $w \in \mathcal{S}$:

$$\begin{aligned} p_w(t+1) &= \sum_{w''' \in \mathcal{S}} P\left(w''' \mid \pi(w), w\right) \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(P_{\pi|_{\mathcal{S}}}\right)^{t-1} (w''',w') \cdot P\left(w'' \mid \pi(w'), w'\right) \\ &= \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(P_{\pi|_{\mathcal{S}}}\right)^{t} (w,w') \cdot P\left(w'' \mid \pi(w'), w'\right) \quad \text{car } w \in \mathcal{S} \end{aligned}$$

ce qui termine la récurrence.

Montrons alors que la série $\sum_{t=1}^{T} p_w(t)$ converge :

$$\begin{aligned} \forall\, T > 0\ ,\ \sum_{t=1}^{T} p_w(t) &= \sum_{t=1}^{T} \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(P_{\pi|_{\mathcal{S}}}\right)^{t-1} (w,w') \cdot P\left(w'' \mid \pi(w'), w'\right) \\ &= \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(\sum_{t=1}^{T} \left(P_{\pi|_{\mathcal{S}}}\right)^{t-1} (w,w')\right) \cdot P\left(w'' \mid \pi(w'), w'\right) \end{aligned}$$

Or, $\left(P_{\pi|_{\mathcal{S}}}\right)^{t-1}$ est la matrice de transitions restreintes à l'ensemble des états de $\mathcal{W}\setminus\mathcal{B}$ qui mènent à $\mathcal{B}$ avec une probabilité non nulle en un nombre fini de coups. Ainsi, il existe $T_m > 0$ tel que :

$$\forall\, t > T_m\ ,\ \forall\, w \in \mathcal{S}\ ,\ \exists\, w' \in \mathcal{B}\ :\ \left(P_{\pi|_{\mathcal{W}\setminus\mathcal{B}}}\right)^{t-1} (w,w') > 0$$

Par conséquent, toutes les lignes des matrices de transition en plus de T_m coups réduites à $\mathcal{S}$ ont des **coefficients strictement inférieurs à 1** puisque, sur chaque ligne, des coefficients strictement positifs correspondant aux transitions vers $\mathcal{B}$ en T_m coups ont été enlevés. La norme de ces matrices est donc strictement inférieure à 1, si bien que celle de la matrice $P_{\pi|_\mathcal{S}}$ est nécessairement strictement inférieure à 1. Ainsi, la matrice $\mathbf{I}_\mathcal{S} - P_{\pi|_\mathcal{S}}$ est inversible et la série $\left(\sum_{t=1}^{T} \left(P_{\pi|_\mathcal{S}}\right)^{t-1}\right)_{T\in\mathbb{N}^*}$ converge. Si $\mathcal{S}$ est connu, l'expression de la probabilité d'atteindre $\mathcal{B}$ depuis $\mathcal{W}$ est :

$$\boxed{P = M_{\mathcal{B}\cap\mathcal{W}} + M_\mathcal{S} \cdot \left(I_\mathcal{S} - P_{\pi|_\mathcal{S}}\right)^{-1} \cdot P_\pi \cdot V_\mathcal{B}} \qquad \text{où :}$$

- $V_\mathcal{B}$ est le vecteur de composantes 1 sur $\mathcal{B}$ et 0 ailleurs,
- $M_\mathcal{S}$ est la matrice qui vaut 1 sur les éléments de la diagonale correspondant aux états de $\mathcal{S}$ et 0 ailleurs,
- $M_{\mathcal{B}\cap\mathcal{W}}$ est la matrice qui vaut 1 sur les éléments de la diagonale correspondant aux états de $\mathcal{B} \cap \mathcal{W}$ et 0 ailleurs.

□

Politique de plus sûr chemin stochastique

Puisque la probabilité d'atteindre les sous-buts existe pour chaque politique π, nous définissons la politique de plus sûr chemin stochastique comme étant celle qui maximise la probabilité d'atteindre les sous-buts parmi toutes les politiques déterministes (cf. définition 13)[1]

Caractérisation de la politique de plus sûr chemin stochastique

Comme pour le théorème de convergence de la fonction de valeur des MDP en horizon infini [14], dont nous nous sommes inspirés, nous démontrons dans le théorème 12 que la politique de plus sûr chemin stochastique est la solution itérative d'une équation de programmation dynamique. Toutefois, la démonstration de notre théorème est un peu plus ardue en raison de l'absence du coefficient d'actualisation γ (inutile dans notre cas) : la convergence de notre opérateur de programmation dynamique repose sur l'extraction de sous-matrices inversibles des matrices de transitions $(P_a)_{a\in\mathcal{A}}$, dans un sous-espace d'états $\mathcal{S}$ où la probabilité d'atteindre $\mathcal{B}$ est strictement positive sur tous les états ($\mathcal{S}$ n'a aucun état absorbant).

Notre équation de programmation dynamique converge donc dans tous les cas. Cependant, contrairement à l'opérateur de Bellman [15, 14] qui permet de calculer

[1] Nous pourrions démontrer, comme pour l'équation de Bellman [14], qu'il existe nécessairement une politique déterministe qui maximise la probabilité d'atteindre les sous-buts parmi toutes les politiques histoire-aléatoire dépendantes qui maximisent cette probabilité (cas général).

Définition 13 Politique de plus sûr chemin stochastique

Soit $\mathcal{B}$ un sous-ensemble d'états buts de $\mathcal{V}$.
Soit $\mathcal{W}$ un sous-ensemble d'états atteignables de $\mathcal{V}$.
Soit $\mathcal{S}$ le sous-ensemble des états de $\mathcal{W}$ pour lesquels il existe au moins un chemin de probabilité non nulle menant à $\mathcal{B}$.
La politique π^* de plus sûr chemin stochastique maximise la probabilité d'atteindre $\mathcal{B}$ depuis n'importe quel état de $\mathcal{W}$ au bout d'un temps infini :

$$\forall\, w \in \mathcal{W} \setminus \mathcal{B}\ ,\ \pi^*(w) = \underset{\pi \in \mathcal{F}(\mathcal{W},\mathcal{A})}{\operatorname{argmax}} \left\{ \chi_{\mathcal{S}}(w) \cdot \sum_{(w',w'') \in \mathcal{S} \times \mathcal{B}} \left(\mathbf{I}_{\mathcal{S}} - P_{\pi|_{\mathcal{S}}}\right)^{-1}(w,w') \cdot P_{\pi}(w',w'') \right\}$$

la fonction de valeur optimale, la limite de la distribution de probabilités dépend de la distribution initiale. Ceci est intuitif car la distribution de probabilité initiale de notre opérateur ($\chi_{\mathcal{B}\cap\mathcal{W}}$) correspond aux récompenses du MDP dans l'opérateur de Bellman (distribution de fonction de valeur initiale). Ces termes ont d'ailleurs le même rôle algébrique dans les deux opérateurs de programmation dynamique.

Théorème 12 *Caractérisation de la politique de plus sûr chemin stochastique*

Soit $\mathcal{B}$ un sous-ensemble d'états buts de $\mathcal{V}$.
Soit $\mathcal{W}$ un sous-ensemble d'états atteignables de $\mathcal{V}$.
La politique π^* de plus sûr chemin stochastique vérifie :

$$\forall\, w \in \mathcal{W} \setminus \mathcal{B}\ ,\ \pi^*(w) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left\{ \sum_{w' \in \mathcal{W}} P(w' \mid a, w) \cdot P^*_{w'} \right\}$$

où $(P^*_w)_{w\in\mathcal{W}}$ est l'unique point fixe de l'équation de programmation dynamique :
– $\forall\, w \in \mathcal{W}\ ,\ P(0) = \chi_{\mathcal{B}\cap\mathcal{W}}$
– $\forall\, t > 0\ ,\ \forall\, w \in \mathcal{W}\ ,$

$$P_w(t) = \chi_{\mathcal{B}\cap\mathcal{W}}(w) + \chi_{\mathcal{W}\setminus\mathcal{B}}(w) \cdot \left(\max_{a \in \mathcal{A}} \left\{ \sum_{w' \in \mathcal{W}} P(w' \mid a, w) \cdot P_{w'}(t-1) \right\} \right)$$

Démonstration. Soit L l'opérateur de programmation dynamique :

$$L\ :\ \left\{ \begin{array}{ccc} \mathbb{R}^{|\mathcal{W}|} & \longrightarrow & \mathbb{R}^{|\mathcal{W}|} \\ P & \longmapsto & \left[\chi_{\mathcal{B}\cap\mathcal{W}}(w) + \chi_{\mathcal{W}\setminus\mathcal{B}}(w) \cdot \left(\max_{a \in \mathcal{A}} \left\{ \sum_{w' \in \mathcal{W}} P(w' \mid a, w) \cdot P_{w'} \right\} \right) \right]_{w \in \mathcal{W}} \end{array} \right.$$

Soit $(P_t)_{t\geqslant 0}$ une suite d'éléments de $\mathcal{W}$ tels que :
- $\forall\, w \in \mathcal{W}\ ,\ P(0) = \chi_{\mathcal{B}\cap\mathcal{W}}$,
- $\forall\, t > 0\ ,\ P(t) = L \cdot P(t-1)$.

Soit $\mathcal{S}$ l'ensemble des états de $\mathcal{W}$ pour lesquels il existe un chemin de probabilité non nulle menant à $\mathcal{B}$.
Montrons que l'opérateur L n'est évalué en fait que sur $\mathcal{S}$, c'est-à-dire :

$$\forall\, t > 0\ , \forall\, w \in \mathcal{W}\ ,\ P_w(t) = \chi_{\mathcal{B}\cap\mathcal{W}}(w) + \chi_{\mathcal{S}}(w)\cdot \left(\max_{a\in\mathcal{A}}\left\{\sum_{w'\in\mathcal{S}} P_{|_{\mathcal{S}}}(w' \mid a, w)\cdot P_{w'}(t-1) + \sum_{w'\in\mathcal{B}\cap\mathcal{W}} P(w' \mid a, w)\right\}\right)$$

Ceci revient à démontrer que la suite $(P_t)_{t\geqslant 0}$ est nulle sur $(\mathcal{W}\setminus B)\setminus\mathcal{S}$: Soit $w \in (\mathcal{W}\setminus B)\setminus\mathcal{S}$.
- $P(0) = \chi_{\mathcal{B}\cap\mathcal{W}}$ donc $P(0)$ est nulle sur $(\mathcal{W}\setminus B)\setminus\mathcal{S}$ car $(\mathcal{W}\setminus\mathcal{B})\cap\mathcal{B} = \emptyset$
- Si $t > 0$:

$$\begin{aligned}
P_w(t) &= \max_{a\in\mathcal{A}}\left\{\sum_{w'\in\mathcal{W}} P(w' \mid a, w)\cdot P_{w'}(t-1)\right\}\\
&= \sum_{w'\in\mathcal{W}} P(w' \mid a_t, w)\cdot P_{w'}(t-1) \qquad [\text{maximum atteint en } a_t]\\
&= \sum_{w'\in\mathcal{W}} P(w' \mid a_t, w)\cdot\left(\max_{a\in\mathcal{A}}\left\{\sum_{w''\in\mathcal{W}} P(w'' \mid a, w')\cdot P_{w''}(t-2)\right\}\right)\\
&= \cdots\\
&= \sum_{(w_1,\ldots,w_t)\in\mathcal{W}^t}\left(\prod_{i=1}^{t} P(w_{i-1} \mid a_i, w_i)\right) P_{w_0}(0)\\
&= \sum_{(w_1,\ldots,w_t)\in\mathcal{W}^t} P(w_0,\ldots,w_t \mid a_0,\ldots,a_t) P_{w_0}(0)
\end{aligned}$$

Si la récurrence ne descend pas jusqu'au rang 0 dans une des sommes, cela signifierait qu'un des états $(w_i)_{1\leqslant i\leqslant t}$ est dans $\mathcal{B}$, ce qui est exclu puisque $w \notin \mathcal{S}$. Discutons sur w_0 :
- Si $w_0 \in \mathcal{B}$ alors $P(w_0,\ldots,w_t \mid a_0,\ldots,a_t)$ est nécessairement nul pour tout $(w_1,\ldots,w_t) \in \mathcal{W}^t$ car $w \notin \mathcal{S}$.
- Si $w_0 \notin \mathcal{B}$ alors $P_{w_0}(0) = 0$.

Donc, dans tous les cas, $P(t)$ est nul sur $(\mathcal{W}\setminus B)\setminus\mathcal{S}$.
Par conséquent :

$$
\begin{aligned}
P_w(t) &= \chi_{\mathcal{B}\cap\mathcal{W}}(w) + \chi_{\mathcal{W}\setminus\mathcal{B}}(w)\cdot\left(\max_{a\in\mathcal{A}}\left\{\sum_{w'\in\mathcal{W}} P(w' \mid a,w)\cdot P_{w'}(t-1)\right\}\right)\\
&= \chi_{\mathcal{B}\cap\mathcal{W}}(w) + \chi_{\mathcal{S}}(w)\cdot\left(\max_{a\in\mathcal{A}}\left\{\sum_{w'\in\mathcal{W}} P(w' \mid a,w)\cdot P_{w'}(t-1)\right\}\right)\\
&= \chi_{\mathcal{B}\cap\mathcal{W}}(w) + \chi_{\mathcal{S}}(w)\cdot\left(\max_{a\in\mathcal{A}}\left\{\sum_{w'\in\mathcal{S}} P_{|_{\mathcal{S}}}(w' \mid a,w)\cdot P_{w'}(t-1) + \sum_{w'\in\mathcal{B}\cap\mathcal{W}} P(w' \mid a,w)\right\}\right)
\end{aligned}
$$

En effet, $P(t-1)=1$ sur $\mathcal{B}\cap\mathcal{W}$.
Ce résultat intermédiaire va nous permettre de faire le lien avec le théorème 11, et en particulier de démontrer la convergence de notre opérateur de programmation dynamique L.
Commençons par montrer que L est contractant sur $\mathcal{S}$ (le point fixe est évident sur le complémentaire de $\mathcal{S}$).
Soient P et U deux éléments de $\mathbb{R}^{|\mathcal{W}|}$, et $w\in\mathcal{S}$. Soit $t>0$.
Supposons que $L\cdot P_w \geqslant L\cdot U_w$.
Soit a_w l'action qui maximise $L\cdot P_w$. A priori, cette action ne maximise pas $L\cdot U_w$, c'est-à-dire :

$$
L\cdot U_w \geqslant \sum_{w'\in\mathcal{S}} P_{|_{\mathcal{S}}}(w' \mid a_w,w)U_{w'} + \sum_{w'\in\mathcal{B}\cap\mathcal{W}} P(w' \mid a_w,w)
$$

$$
\begin{aligned}
|L\cdot P_w - L\cdot U_w| &= L\cdot P_w - L\cdot U_w\\
&\leqslant \sum_{w'\in\mathcal{S}} P_{|_{\mathcal{S}}}(w' \mid a^*_w,w)\cdot(P_{w'}-U_{w'})\\
&\leqslant \sum_{w'\in\mathcal{S}} P_{|_{\mathcal{S}}}(w' \mid a^*_w,w)\cdot\|P-U\|
\end{aligned}
$$

Or, nous avons démontré dans le théorème 11 que la norme de la matrice $P_{|_{\mathcal{S}}}$ est strictement inférieure à 1, ce qui signifie que tous ses coefficients sont strictement inférieurs à 1. Comme la somme des éléments de chaque ligne est inférieure ou égale à 1 (en tant que restriction d'une matrice stochastique), nous en déduisons que la somme des éléments de chaque ligne est strictement inférieure à 1 :

$$
\forall\, w\in\mathcal{S}\ ,\ \sum_{w'\in\mathcal{S}} P_{|_{\mathcal{S}}}(w' \mid a^*_w,w) < 1
$$

Aussi, le maximum M des sommes précédentes sur toutes les lignes est strictement inférieur à 1 :

$$
M = \max_{w\in\mathcal{S}}\left\{\sum_{w'\in\mathcal{S}} P_{|_{\mathcal{S}}}(w' \mid a^*_w,w)\right\} < 1
$$

Ainsi :

$$\forall\, w \in \mathcal{S}\ ,\ |L \cdot P_w - L \cdot U_w| = L \cdot P_w - L \cdot U_w \leqslant M\|P - U\|$$

Donc : $\|P - U\| \leqslant M\|P - U\|$ avec $M < 1$, ce qui signifie que L est contractante. D'après le théorème du point fixe de Banach, nous en déduisons que L admet un unique point fixe.

Montrons que cet unique point fixe est la probabilité d'atteindre les sous-buts en suivant la politique π^* de plus sûr chemin stochastique, soit P^*.
Soit $P \in \mathbb{R}^{|\mathcal{W}|}$ tel que : $P \geqslant L \cdot P$.
Si $w \in \mathcal{B} \cap \mathcal{W} : P_w \geqslant (L \cdot P)(w) = 1 = P^*_w$ (d'après le théorème 11).
Si $w \in (\mathcal{W} \setminus \mathcal{B}) \setminus \mathcal{S} : P_w \geqslant (L \cdot P)(w) = 0 = P^*_w$ (d'après le théorème 11).
Si $w \in \mathcal{S}$:

$$\begin{aligned}
P_w &\geqslant (L \cdot P)(w) \\
&\geqslant \max_{a \in \mathcal{A}} \left\{ \sum_{w' \in \mathcal{S}} P_{|\mathcal{S}}(w' \mid a, w) \cdot P_{w'} + \sum_{w' \in \mathcal{B} \cap \mathcal{W}} P(w' \mid a, w) \right\} \\
&\geqslant \sum_{w' \in \mathcal{S}} P_{|\mathcal{S}}(w' \mid \pi^*(w), w) \cdot P_{w'} + \sum_{w' \in \mathcal{B} \cap \mathcal{W}} P(w' \mid \pi^*(w), w) \\
&\geqslant \sum_{w' \in \mathcal{S}} P_{|\mathcal{S}}(w' \mid \pi^*(w), w) \left(\sum_{w'' \in \mathcal{S}} P_{|\mathcal{S}}(w'' \mid \pi^*(w'), w') \cdot P_{w''} \right. \\
&\qquad \left. + \sum_{w'' \in \mathcal{B} \cap \mathcal{W}} P(w'' \mid \pi^*(w'), w') \right) + \sum_{w' \in \mathcal{B} \cap \mathcal{W}} P(w' \mid \pi^*(w), w) \\
&\geqslant \cdots \\
&\geqslant \sum_{t=1}^{T} \sum_{(w_0, \ldots, w_T) \in \mathcal{S}^T \times \mathcal{B}} \left(\prod_{i=1}^{t-1} P_{|\mathcal{S}}(w_i \mid \pi^*(w_{i-1}), w_{i-1}) \right) P(w_T \mid \pi^*(w_{T-1}), w_{T-1}) \\
&\qquad + \sum_{(w_0, \ldots, w_{T-1}) \in \mathcal{S}^T} \left(\prod_{i=1}^{T} P_{|\mathcal{S}}(w_i \mid \pi^*(w_{i-1}), w_{i-1}) \right) P_{w_T} \\
&\geqslant \sum_{t=1}^{T} \sum_{(w', w'') \in \mathcal{S} \times \mathcal{B}} \left(P_{\pi^*|\mathcal{S}} \right)^{t-1} (w, w') P_{\pi^*}(w', w'') + \sum_{w' \in \mathcal{S}} \left(P_{\pi^*|\mathcal{S}} \right)^{T} (w, w') P_{w'}
\end{aligned}$$

Or, d'après le théorème 11, si $w \in \mathcal{S}$:

$$\begin{aligned} P^*_w &= \sum_{(w',w'')\in\mathcal{S}\times\mathcal{B}} \left(\mathbf{I}_{\mathcal{S}} - P_{\pi^*|_{\mathcal{S}}}\right)^{-1}(w,w') \cdot P_{\pi^*}(w',w'') \\ &= \sum_{t=1}^{+\infty} \sum_{(w',w'')\in\mathcal{S}\times\mathcal{B}} \left(P_{\pi^*|_{\mathcal{S}}}\right)^{t-1}(w,w') \cdot P_{\pi^*}(w',w'') \end{aligned}$$

Par conséquent :

$$P_w - P^*_w \geqslant \sum_{t=T+1}^{+\infty} \sum_{(w',w'')\in\mathcal{S}\times\mathcal{B}} \left(P_{\pi^*|_{\mathcal{S}}}\right)^{t-1}(w,w') \cdot P_{\pi^*}(w',w'') + \sum_{w'\in\mathcal{S}} \left(P_{\pi^*|_{\mathcal{S}}}\right)^{T}(w,w') P_{w'}$$

Or les deux termes précédents tendent vers 0 quand T tend vers l'infini, car la norme de la matrice $P_{\pi^*|_{\mathcal{S}}}$ est strictement inférieure à 1.
Ainsi, dans tous les cas : $P \geqslant L \cdot P \Longrightarrow P \geqslant P^* \qquad (i)$
Supposons maintenant que $P \leqslant L \cdot P$.
Si $w \in \mathcal{B} \cap \mathcal{W}$: $P_w \leqslant (L \cdot P)(w) = 1 = P^*_w$ (d'après le théorème 11).
Si $w \in (\mathcal{W} \setminus \mathcal{B}) \setminus \mathcal{S}$: $P_w \leqslant (L \cdot P)(w) = 0 = P^*_w$ (d'après le théorème 11).
Si $w \in \mathcal{S}$:
Soit la politique :

$$\overline{\pi} \ : \ \left\{ \begin{array}{ccl} \mathcal{S} & \to & \mathcal{A} \\ w & \mapsto & \underset{a\in\mathcal{A}}{\operatorname{argmax}} \left\{ \sum_{w'\in\mathcal{S}} P_{|\mathcal{S}}(w' \mid a, w) \cdot P_{w'} + \sum_{w'\in\mathcal{B}\cap\mathcal{W}} P(w' \mid a, w) \right\} \end{array} \right.$$

Alors :

$$\begin{aligned} P_w &\leqslant (L \cdot P)(w) \\ &\leqslant \max_{a\in\mathcal{A}} \left\{ \sum_{w'\in\mathcal{S}} P_{|\mathcal{S}}(w' \mid a, w) \cdot P_{w'} + \sum_{w'\in\mathcal{B}\cap\mathcal{W}} P(w' \mid a, w) \right\} \\ &\leqslant \sum_{w'\in\mathcal{S}} P_{|\mathcal{S}}(w' \mid \overline{\pi}(w), w) \cdot P_{w'} + \sum_{w'\in\mathcal{B}\cap\mathcal{W}} P(w' \mid \overline{\pi}(w), w) \\ &\leqslant \cdots \\ &\leqslant \sum_{t=1}^{T} \sum_{(w',w'')\in\mathcal{S}\times\mathcal{B}} \left(P_{\overline{\pi}|_{\mathcal{S}}}\right)^{t-1}(w,w') P_{\overline{\pi}}(w',w'') + \sum_{w'\in\mathcal{S}} \left(P_{\overline{\pi}|_{\mathcal{S}}}\right)^{T}(w,w') P_{w'} \end{aligned}$$

Or, d'après la définition 13, si $w \in \mathcal{S}$:

$$\begin{aligned} P^*_w &= \max_{\pi \in \mathcal{F}(\mathcal{S},\mathcal{A})} \left\{ \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(\mathbf{I}_\mathcal{S} - P_{\pi|_\mathcal{S}}\right)^{-1}(w,w') \cdot P_\pi(w',w'') \right\} \\ &\geqslant \sum_{t=1}^{+\infty} \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(P_{\overline{\pi}|_\mathcal{S}}\right)^{t-1}(w,w') \cdot P_{\overline{\pi}}(w',w'') \end{aligned}$$

Par conséquent :

$$P_w - P^* \leqslant \sum_{t=T+1}^{+\infty} \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(P_{\overline{\pi}|_\mathcal{S}}\right)^{t-1}(w,w') \cdot P_{\overline{\pi}}(w',w'') + \sum_{w' \in \mathcal{S}} \left(P_{\overline{\pi}|_\mathcal{S}}\right)^{T}(w,w') P_{w'}$$

Or les deux termes précédents tendent vers 0 quand T tend vers l'infini, car la norme de la matrice $P_{\overline{\pi}|_\mathcal{S}}$ est strictement inférieure à 1.
Ainsi, dans tous les cas : $P \leqslant L \cdot P \Longrightarrow P \leqslant P^* \qquad (ii)$
D'après (i) et (ii) : $P = L \cdot P \Longrightarrow P = P^*$, ce qui prouve que P^* est l'unique point fixe de L.
Il reste à démontrer que :

$$\forall\ w \in \mathcal{W} \setminus \mathcal{B}\ ,\ \pi^*(w) = \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ \sum_{w' \in \mathcal{W}} P(w' \mid a, w) \cdot P^*_{w'} \right\}$$

Si $w \in (\mathcal{W} \setminus \mathcal{B}) \setminus \mathcal{S}$, d'après la définition 13 : $\pi^*(w) = \operatorname*{argmax}_{\pi\mathcal{F}(\mathcal{W},\mathcal{A})}\{0\}$.
Or P^* est point fixe de L donc, d'après la définition de cet opérateur, P^* est nul sur $(\mathcal{W} \setminus \mathcal{B}) \setminus \mathcal{S}$. Ainsi : $\max\limits_{a \in \mathcal{A}} \sum\limits_{w' \in \mathcal{W}} P(w' \mid a, w) \cdot P^*_{w'} = P^*_w = 0$
Si $w \in \mathcal{S}$, d'après la définition 13 :

$$\begin{aligned} \pi^*_w &= \operatorname*{argmax}_{\pi \in \mathcal{F}(\mathcal{S},\mathcal{A})} \left\{ \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(\mathbf{I}_\mathcal{S} - P_{\pi|_\mathcal{S}}\right)^{-1}(w,w') \cdot P_\pi(w',w'') \right\} \\ &= \operatorname*{argmax}_{\pi \in \mathcal{F}(\mathcal{S},\mathcal{A})} \left\{ \sum_{t=1}^{+\infty} \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(P_{\pi|_\mathcal{S}}\right)^{t-1}(w,w') \cdot P_\pi(w',w'') \right\} \end{aligned}$$

$$
\begin{aligned}
&= \operatorname*{argmax}_{\pi \in \mathcal{F}(\mathcal{S},\mathcal{A})} \left\{ \underbrace{\sum_{w'' \in \mathcal{B}} P_\pi(w, w'')}_{w'=w \text{ en 1 coup}} + \sum_{t=2}^{+\infty} \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(P_{\pi|_\mathcal{S}}\right)^{t-1}(w, w') \cdot P_\pi(w', w'') \right\} \\
&= \operatorname*{argmax}_{\pi \in \mathcal{F}(\mathcal{S},\mathcal{A})} \left\{ \sum_{\tilde{w} \in \mathcal{W}} P_\pi(w, \tilde{w}) \cdot \chi_{\mathcal{B}\cap\mathcal{W}}(\tilde{w}) \right. \\
&\qquad \left. + \sum_{\tilde{w} \in \mathcal{S}} P_\pi(w, \tilde{w}) \sum_{t=2}^{+\infty} \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(P_{\pi|_\mathcal{S}}\right)^{t-2}(\tilde{w}, w') \cdot P_\pi(w', w'') \right\} \\
&= \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ \sum_{\tilde{w} \in \mathcal{W}} P_a(w, \tilde{w}) \cdot \left(\chi_{\mathcal{B}\cap\mathcal{W}}(\tilde{w}) \right.\right. \\
&\qquad \left.\left. + \chi_\mathcal{S}(\tilde{w}) \cdot \max_{\pi \in \mathcal{F}(\mathcal{S},\mathcal{A})} \left\{ \sum_{(w',w'') \in \mathcal{S}\times\mathcal{B}} \left(\mathbf{I}_\mathcal{S} - P_{\pi|_\mathcal{S}}\right)^{-1}(\tilde{w}, w') \cdot P_\pi(w', w'') \right\} \right) \right\} \\
&= \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ \sum_{\tilde{w} \in \mathcal{W}} (\tilde{w} \mid a, w) \cdot P^*_{\tilde{w}} \right\} \qquad \text{[d'après la définition 13]}
\end{aligned}
$$

ce qui termine la démonstration en prenant $w' = \tilde{w}$. □

8.3.2 Algorithme

L'algorithme 30, qui permet de calculer la politique de plus sûr chemin stochastique, utilise l'opérateur de programmation dynamique défini dans le théorème 12. L'existence de cette politique, purement mathématique, ne dépend pas du modèle informatique utilisé. Ainsi, bien que notre algorithme concerne des MDP factorisés, nous pourrions tout autant proposer un formalisme en espace d'états énumérés. La version énumérée de notre algorithme est la même que celle d'itération de la valeur ou de la politique des MDP énumérés en remplaçant l'opérateur de Bellman par notre opérateur de programmation dynamique [14].

L'algorithme 30 se découpe en 4 étapes principales :

1. Calcul des BDD et des ADD qui masquent la politique et les transitions sur $\mathcal{B} \cap \mathcal{W}$ et $\mathcal{W} \setminus \mathcal{B}$: ces masques représentent les fonctions caractéristiques $\chi_{\mathcal{B}\cap\mathcal{W}}$ et $\chi_{\mathcal{W}\setminus\mathcal{B}}$ de notre opérateur de programmation dynamique (cf. théorème 12) ;
2. restriction des transitions de l'ensemble de départ $\mathcal{W} \setminus \mathcal{B}$ vers l'ensemble d'arrivée $\mathcal{W}$: en effet, d'après le théorème 12, les transitions $P(w' \mid a, w)$ ne sont évaluées que pour $w \in \mathcal{W} \setminus \mathcal{B}$ et $w' \in \mathcal{W}$;

3. calcul itératif de P^* à ϵ près ;
4. calcul de la politique restreinte à $\mathcal{W} \setminus \mathcal{B}$ sur la base du P^* précédemment calculé (cf. théorème 12.

La politique, en tant que liste de BDD représentant pour chaque action l'ensemble d'états où doit être appliquée l'action, est calculée de la même manière que pour la politique qui maximise une fonction de valeur factorisée (cf. algorithme 28). La différence entre le maximum courant et la probabilité d'atteindre les sous-buts avec l'action courante est un ADD qui est nul sur les états où cette action est la meilleure. La transformation de cet ADD en BDD, puis le calcul du complémentaire du BDD obtenu à l'intérieur de l'espace de recherche ($\mathcal{W} \setminus \mathcal{B}$), permettent d'obtenir le BDD où cette action doit être appliquée.

En revanche, contrairement à la programmation dynamique classique, l'ADD représentant le maximum de la probabilité P sur toutes les actions n'a pas besoin d'être initialisé pour la première action. En effet, P est toujours positif et P est initialisé à la valeur 0, si bien que la référence du maximum est implicitement la première politique, dont la valeur est nécessairement plus grande que 0.

La complexité de notre algorithme itératif de calcul du plus sûr chemin stochastique est du même ordre de grandeur que celui du calcul de la fonction de valeur optimale d'un MDP. En effet, dans notre opérateur de programmation dynamique, l'équivalent du coefficient γ de l'opérateur de Bellman est le coefficient maximum de la matrice $P_{\pi^*|_{\mathcal{S}}}$, soit M. Par transposition, la complexité de notre algorithme est donc [14] :

$$\boxed{O\left(\frac{\ln \epsilon(1-M)}{\ln M}|\mathcal{A}||\mathcal{W} \setminus \mathcal{B}|^2\right)}$$

8.3.3 Comparaison entre l'heuristique de plus sûr chemin stochastique et l'heuristique de sLAO*

Cette heuristique a deux avantages majeurs sur une heuristique de plus court chemin stochastique :

- certes elle trouve des chemins plus longs (et donc des espaces atteignables plus grands), mais elle maximise la probabilité de réussite de la mission (ce qui est plus important que de terminer rapidement la mission),
- elle ne modifie pas le modèle de transitions (inutile de rajouter -1 à chaque action), ce qui permet de gérer des ADD beaucoup plus petits durant le calcul de la politique.

Toutefois, si le modèle l'impose, il est possible de combiner une heuristique de plus sûr chemin stochastique et une autre de plus court chemin stochastique : soit lorsque chaque transition coûte -1, soit lorsqu'une variable d'état `autonomie` de valeur

Algorithme 30 : Fonction `PlusSurCheminStochastique`

Données : $\mathcal{B}$ (BDD), $\mathcal{W}$ (BDD)
Résultat : P (ADD), π (liste de BDD)
début
  $\mathcal{B}and\mathcal{W} \leftarrow$ `bddAnd`$(\mathcal{B}, \mathcal{W})$;
  $add\mathcal{B}and\mathcal{W} \leftarrow$ `BddToAdd`$(\mathcal{B}and\mathcal{W})$;
  $\mathcal{W}minus\mathcal{B} \leftarrow$ `bddXor`$(\mathcal{W}, \mathcal{B}and\mathcal{W})$;
  $add\mathcal{W}minus\mathcal{B} \leftarrow$ `BddToAdd`$(\mathcal{W}minus\mathcal{B})$;
  $add\mathcal{W}primed \leftarrow$ `BddToAdd`(`bddPermute`$(\mathcal{W}, unprimed\_to\_primed)$);
  pour $a \in \mathcal{A}$ **faire**
    $T^a_{|\mathcal{W}\setminus\mathcal{B}} \leftarrow$ `addApply`$(\texttt{addTimes}, T^a, add\mathcal{W}minus\mathcal{B})$;
    $T^a_{|(\mathcal{W}\setminus\mathcal{B},\mathcal{W}')} \leftarrow$ `addApply`$(\texttt{addTimes}, T^a_{|\mathcal{W}\setminus\mathcal{B}}, add\mathcal{W}primed)$;
  $primed\_variables \leftarrow$ `addComputeCube` $\left(\wedge_{1\leqslant i\leqslant|\mathcal{V}|}\{X'_i\}\right)$;
  $P \leftarrow add\mathcal{B}and\mathcal{W}$;
  répéter
    $saveP \leftarrow P$;
    $savePprimed \leftarrow$ `addPermute`$(saveP, unprimed\_to\_primed)$;
    $P \leftarrow$ `ReadZero`$()$;
    pour $a \in \mathcal{A}$ **faire**
      $Ptemp \leftarrow$ `addApply`$(\texttt{addTimes}, T^a_{|(\mathcal{W}\setminus\mathcal{B},\mathcal{W}')}, savePprimed)$;
      $Ptemp \leftarrow$ `addExistAbstract`$(Ptemp, primed\_variables)$;
      $P \leftarrow$ `addApply`$(\texttt{addMaximum}, P, Ptemp)$;
    $P \leftarrow$ `addApply`$(\texttt{addPlus}, P, add\mathcal{B}and\mathcal{W})$;
  jusqu'à `EqualSupNorm`$(P, saveP, \epsilon)$ = **true**;
  $P \leftarrow$ `ReadZero`$()$;
  pour $a \in \mathcal{A}$ **faire**
    $Ptemp \leftarrow$ `addApply`$(\texttt{addTimes}, T^a_{|(\mathcal{W}\setminus\mathcal{B},\mathcal{W}')}, savePprimed)$;
    $Ptemp \leftarrow$ `addExistAbstract`$(Ptemp, primed\_variables)$;
    $P \leftarrow$ `addApply`$(\texttt{addMaximum}, P, Ptemp)$;
    si $a = \mathcal{A}.debut()$ **alors**
      $\pi[a] \leftarrow \mathcal{W}minus\mathcal{B}$;
    sinon
      $diff \leftarrow$ `addApply`$(\texttt{addMinus}, P, Ptemp)$;
      $\pi[a] \leftarrow$
      `bddXor`$(\mathcal{W}minus\mathcal{B},$ `bddAnd`$($`addBddPattern`$(diff), \mathcal{W}minus\mathcal{B}))$;
      pour $a' = \mathcal{A}.debut()$ **jusqu'à** $a' = \mathcal{A}.precedent(a)$ **faire**
        $\pi[a'] \leftarrow$ `bddXor`$(\pi[a'],$ `bddAnd`$(\pi[a'], \pi[a]))$;
  retourner (P, π);
fin

décroissante limite les déplacements dans l'espace d'états. Ce dernier cas est pris en compte dans notre outil de modélisation et de résolution de missions ReSSAC (cf. chapitre 6 page 153).

La complexité de notre algorithme de plus sûr chemin stochastique est meilleure que celle du calcul de l'heuristique utilisée dans `sLAO*`. Cette dernière est en effet une approximation de la fonction de valeur calculée *sur tout l'espace d'états*, donc sa complexité est :

$$O\left(\frac{\ln \tilde{\epsilon}(1-\gamma)}{\ln \gamma}|\mathcal{A}||\mathcal{V}|^2\right) \qquad (\tilde{\epsilon} < \epsilon)$$

La complexité en ϵ est logarithmique alors que celle en espace d'états est polynomiale. Notre heuristique a donc des chances d'être moindre, si toutefois la première phase de son calcul (obtention de $\mathcal{W}$) est suffisamment efficace ($|\mathcal{W} \setminus \mathcal{B}| \ll |\mathcal{V}|$).

De plus, notre heuristique n'utilise pas les récompenses du MDP. Outre l'économie du calcul de la sommation avec les récompenses dans l'opérateur de Bellman, ceci permet de stabiliser le calcul itératif de la valeur optimale, et donc parfois de l'accélérer. Néanmoins, une heuristique peut être catastrophique si elle guide mal la recherche de la solution optimale (avec récompenses). Ce cas peut bien entendu survenir avec notre heuristique, en particulier lorsque de fortes pénalités se trouvent sur les chemins qui mènent le plus sûrement aux sous-buts de la mission : l'itération de la valeur (ou de la politique) nécessitera plus d'itérations pour corriger l'erreur induite par l'heuristique. Ainsi, **l'heuristique de plus sûr chemin stochastique ne doit pas être utilisée si l'optimisation du gain est plus importante que la probabilité d'atteindre les sous-buts de la mission**. Nous discuterons de ce point très important lors des tests dans le chapitre suivant.

8.4 Réduction de l'espace d'états atteignables par filtrage stochastique des états

8.4.1 Utilisation de la politique de plus sûr chemin stochastique

L'espace d'états atteignables initialement calculé peut être réduit grâce à la donnée de la plus forte probabilité d'atteindre les sous-buts en chaque état de $\mathcal{W}$. Le premier espace d'états atteignables a été obtenu en propageant les effets déterministes des actions depuis les états initiaux possibles $\mathcal{I}$, jusqu'aux sous-buts $\mathcal{B}$. Cette propagation n'étant guidée par aucune politique, l'espace obtenu est isotrope depuis $\mathcal{I}$, comme le montre la figure 8.1 (page 196).

Ainsi, de nombreux états de $\mathcal{W}$ n'ont pratiquement aucune chance d'être sur les chemins de fortes probabilités menant de $\mathcal{I}$ à $\mathcal{B}$: ce sont essentiellement les états

qui se trouvent à l'opposé des trajectoires directes entre $\mathcal{I}$ et $\mathcal{B}$ (cf. figure 8.1 page 196).

Nous proposons donc de profiter du calcul de la probabilité de plus sûr chemin stochastique pour filtrer les états dont la probabilité d'atteindre les sous-buts est faible par rapport aux états de $\mathcal{I}$. L'espace d'états atteignables réduit, soit $\tilde{\mathcal{W}}$, est défini ainsi :

$$\tilde{\mathcal{W}} = \left\{ w \in \mathcal{W} \ : \ P^*_w > \zeta \cdot \left(\min_{i \in \mathcal{I}} P^*_i \right) \right\} \qquad , \quad \zeta \geqslant 0$$

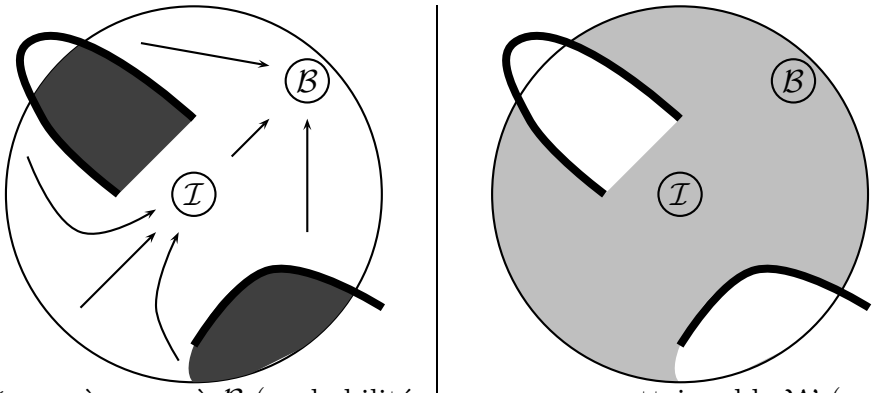
Le coefficient de filtrage ζ doit être assez petit. La probabilité d'atteindre les sous-buts étant une moyenne sur de nombreuses trajectoires menant aux sous-buts, il est en effet possible qu'un état w d'une trajectoire possible de π^* soit tel que : $P^*_i \ll \min_{i \in \mathcal{I}} P^*_i$.

Cependant, un tel état w a de grandes chances d'appartenir à des trajectoires très peu probables de π^*, car sa très faible probabilité d'atteindre $\mathcal{B}$ «plombe» la moyenne des probabilités d'atteindre les sous-buts pour les états parents de ces trajectoires. Aussi, en partant dans $\mathcal{I}$, l'agent a de fortes chances de ne jamais passer en w, mais avec une probabilité toutefois bien plus grande que celle de ne jamais arriver aux sous-buts depuis w. Par conséquent, un filtrage trop large risque de couper de nombreuses trajectoires très probables de π^*, si bien que ζ doit être très petit.

Le cas limite $\zeta = 0$ ne supprime aucune trajectoire menant aux sous-buts en suivant la politique de plus sûr chemin stochastique, car il filtre uniquement les états qui ont une probabilité nulle d'atteindre les sous-buts. Ce cas est néanmoins intéressant pour des problèmes où les actions définies sur certains états atteignables ne permettent pas de «revenir en arrière» : en appliquant toutes les actions possibles depuis les états initiaux, il est possible de se retrouver dans un couloir «aspirant» qui ne mène pas aux buts (toutes les actions ont une probabilité nulle d'atteindre $\mathcal{B}$ dans ce couloir), ou dans un sous-espace séparé des buts par des obstacles (cf. figure 8.3). Ces états atteignables, qui peuvent être très nombreux, sont filtrés pour tout ζ positif ou nul.

8.4.2 Algorithme

Le filtrage stochastique des états de $\mathcal{W}$ nécessite de calculer le minimum de P^* sur $\mathcal{I}$. Cette opération, simple dans une représentation énumérée de l'espace d'états, doit être effectuée avec précaution en version symbolique. En effet, la routine des diagrammes de décision qui retourne le minimum d'une fonction symbolique f, calcule ce minimum sur l'espace d'états entier (du moins, dans la librairie de diagrammes de décision CUDD [70]). Ainsi, même si P^* est réduite à $\mathcal{I}$ grâce à une multiplication de P^* par le masque $\mathcal{I}$, le minimum sera calculé à la fois sur $\mathcal{I}$ **et** sur



π^* ne mène pas à $\mathcal{B}$ (probabilité nulle) sur les zones sombres

sous-espace atteignable $\mathcal{W}$ (zone grisée) après filtrage avec $\zeta = 0$

FIG. 8.3 – Filtrage avec $\zeta = 0$: les états atteignables où aucune action ne mène aux buts — et donc π^* non plus — sont supprimés de $\mathcal{W}$.

le complémentaire nul de $\mathcal{I}$ (cf. figure 8.4). Comme P^* est à valeurs dans $\mathbb{R}^+$, son minimum sera toujours nul, *indépendamment* de ses valeurs.

Néanmoins, un masque de valeur infinie sur le complémentaire de $\mathcal{I}$ assure que le minimum de P^* sur l'espace d'états entier $\mathcal{V}$, combiné à ce masque, est dans $\mathcal{I}$ (cf. figure 8.4) :

$$\min_{w \in \mathcal{I}} P^*_w = \min_{w \in \mathcal{V}} \left((+\infty) \cdot \chi_{\mathcal{V} \setminus \mathcal{I}}(w) + P^*_w \right)$$

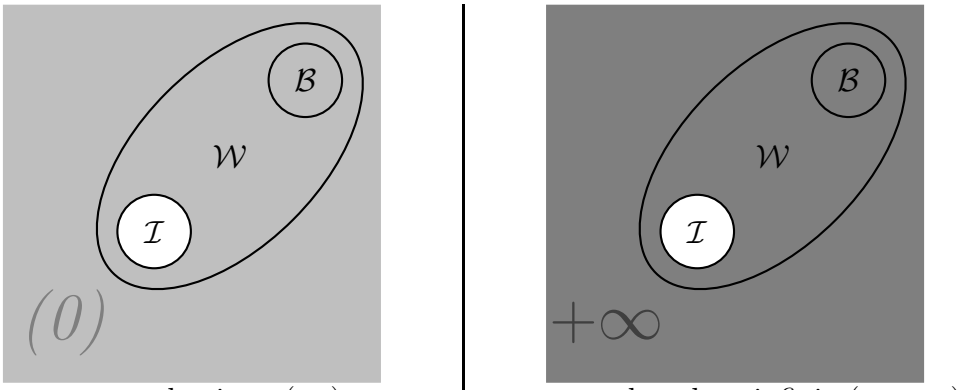
A cette fin, les librairies de diagrammes de décision fournissent une fonction qui renvoie un ADD de valeur uniformément infinie (`ReadPlusInfinity`). Ainsi, nous utilisons cette fonction combinée à l'égalité précédente dans l'algorithme 31, afin d'obtenir un ADD $P^*_{|\mathcal{I}}$ qui vaut $+\infty$ sur $\mathcal{V} \setminus \mathcal{I}$ et P^* sur $\mathcal{I}$.

Notons que le calcul du minimum de P^* est inutile pour $\zeta = 0$. Dans ce cas, la condition de filtrage est simplement $P^* > 0$. Pour des raisons d'efficacité, il convient donc de séparer les cas $\zeta = 0$ (filtrage «déterministe») et $\zeta > 0$ (filtrage «stochastique») dans l'algorithme.

Ainsi, si ζ est non nul, l'algorithme de filtrage stochastique 31 utilise la routine `addBddStrictThreshold`(f, α), qui renvoie un BDD de valeur 1 partout où f est strictement supérieure à α. Le BDD $\tilde{\mathcal{W}}$ ainsi obtenu représente les états de $\mathcal{W}$ où : $P^* > \zeta \cdot min\mathcal{I}$. Si ζ est nul, l'algorithme de filtrage utilise la fonction `addBddPattern`(f), qui renvoie un BDD de valeur 1 partout où f est non nulle.

Remarquons que l'espace d'états atteignables $\tilde{\mathcal{W}}$ est égal à $\mathcal{W}$ si un des états initiaux ne permet pas d'arriver aux sous-buts. La probabilité d'atteindre les sous-buts est alors nulle en cet état si bien que le minimum de P^* sur $\mathcal{I}$ est nul. Aussi, aucun état de $\mathcal{W}$ n'est filtré si bien que $\tilde{\mathcal{W}} = \mathcal{W}$.

Cependant, dans la plupart des problèmes, l'espace d'états est connexe au point



masque classique ($\chi_{\mathcal{I}}$)
$\min_{\mathcal{I}} = 0, 2 \Rightarrow \min_{\mathcal{I} \times masque} = 0$

masque de valeur infinie ($+\infty\chi_{\overline{\mathcal{I}}}$)
$\min_{\mathcal{I}} = 0, 2 \Rightarrow \min_{\mathcal{I}+masque} = 0, 2$

FIG. 8.4 – Nécessité d'utiliser un masque de valeur infinie pour calculer le minimum sur un sous-ensemble d'état d'une fonction symbolique (factorisée)

Algorithme 31 : Fonction `FiltrageStochastique`

Données : $\mathcal{I}$ (BDD), P^* (ADD), $\zeta \geqslant 0$
Résultat : $\tilde{\mathcal{W}}$ (BDD)
début
  si $\zeta = 0$ **alors**
    $\tilde{\mathcal{W}} \leftarrow$ `addBddPattern`(P^*);
  sinon
    $add\overline{\mathcal{I}} \leftarrow$ `BddToAdd`(`bddXor`(`ReadOne`(), $\mathcal{I}$));
    $masque \leftarrow$ `addApply`(`addTimes`, $add\overline{\mathcal{I}}$, `ReadPlusInfinity`());
    $P^*_{|\mathcal{I}} \leftarrow$ `addApply`(`addPlus`, P^*, $masque$);
    $min\mathcal{I} \leftarrow$ `addFindMin`$(P^*_{|\mathcal{I}})$;
    $\tilde{\mathcal{W}} \leftarrow$ `addBddStrictThreshold`$(P^*, \zeta * min\mathcal{I})$;
  retourner $\tilde{\mathcal{W}}$;
fin

que le cas précédent ne se présente généralement pas. Dans le chapitre suivant, nous présenterons des tests effectués sur des problèmes RESSAC, qui montrent que l'espace d'états atteignables est très fortement réduit durant la phase du calcul de notre heuristique. La phase de résolution partielle du MDP, que nous présentons dans le prochain chapitre, alterne une expansion de l'espace d'états atteignables à partir de $\tilde{\mathcal{W}}$ et une optimisation du MDP sur l'espace atteignable courant.

Chapitre 9

Intégration des sous-buts de planification à la programmation dynamique

9.1 sfDP : Programmation Dynamique Symbolique et Focalisée

9.1.1 Un algorithme dérivé, mais éloigné, de sLAO*

Dans ce chapitre, nous proposons un algorithme inspiré de sLAO*, qui optimise le MDP sur un sous-espace d'états atteignables sans cesse recalculé avec la politique optimale courante. Comme sLAO*, notre algorithme alterne donc une phase de programmation dynamique stochastique sur le sous-espace atteignable courant, et une phase d'expansion de ce sous-espace atteignable avec la politique courante (cf. figure 7.4 page 184).

Néanmoins, nous utilisons une heuristique radicalement différente de celle de sLAO*. Alors que l'heuristique de sLAO* est une approximation de la fonction de valeur optimale du MDP, notre heuristique indique la politique de plus sûr chemin stochastique vers les sous-buts imposés par la mission. Ces deux notions sont différentes car toutes les récompenses du problème ne sont pas nécessairement imposées. Si les sous-buts imposés englobe toutes les récompenses, notre algorithme est alors équivalent à sLAO* car nous luis imposons d'atteindre toutes les récompenses du problème. Ainsi, nous distinguons de mode de fonctionnement :

- mode *opportuniste* : les récompenses ne sont pas toutes des sous-buts de la mission ; l'optimisation de la stratégie consiste à atteindre *obligatoirement* toutes les récompenses imposées, en essayant d'en récolter d'autres *si possible*.
- mode *optimal* : *toutes* les récompenses sont des sous-buts imposées par la

mission ; l'optimisation de la stratégie consiste donc à optimiser le problème *global* (comme sLAO*).

Nous avons nommé notre algorithme sfDP («Symbolic Focused Dynamic Programming») car, ainsi que nous l'expliquons dans le paragraphe suivant, **l'heuristique utilisée focalise la politique courante vers les sous-buts sans être informée des récompenses qui se trouvent à l'extérieur de l'espace d'états atteignables**. Le mode optimal étant un cas limite du mode opportuniste, nous montrerons par des expérimentations le comportement asymptotique de sfDP lorsque le nombre de récompenses imposées augmente.

9.1.2 Une découverte opportuniste des récompenses du problème

L'heuristique de sLAO* est *compatible* avec la fonction de valeur du MDP, en ce sens qu'elles ont le même ordre de grandeur et qu'elles indiquent toutes deux la politique à suivre pour maximiser la moyenne des récompenses accumulée durant le processus. Par conséquent, sLAO* utilise cette heuristique, calculée préalablement sur l'espace d'états entier, pour mettre à jour la fonction de valeur sur la frontière de l'espace d'états atteignables courant (cf. figure 7.6 page 188).

Au contraire, sfDP ne peut pas utiliser l'heuristique h de plus sûr chemin stochastique pour mettre à jour la fonction de valeur V du MDP, car elles n'ont ni le même ordre de grandeur ($\mathtt{Im}(h) \subset [0;1]$ et $\mathtt{Im}(V) \subset \mathbb{R}$), ni la même signification. Les politiques de h et V sont corrélées uniquement si les récompenses du MDP sont toutes positives **et** si ces récompenses sont toutes des sous-buts imposés par la mission. Ce cas est le mode optimal, limite du mode opportuniste.

Ainsi, contrairement à sLAO*, l'heuristique de sfDP n'est pas guidée, sur la frontière d'états atteignables, par les récompenses qui se trouvent à l'extérieur des états atteignables. En quelque sorte, notre heuristique est «aveuglée» par les récompenses des sous-buts imposés par la mission, si bien qu'elle ne peut pas «voir» les autres récompenses, situées à l'extérieur de l'espace d'états atteignables.

Cependant, nous donnons l'opportunité à sfDP de découvrir «par chance» les autres récompenses du MDP (cf. figure 9.1) :

- soit lorsqu'elles se trouvent à l'intérieur de l'espace d'états atteignables calculé par l'heuristique,
- soit lorsqu'elles se trouvent sur la frontière de l'espace d'états atteignables.

Notre algorithme est donc clairement sous-optimal en mode opportuniste strict, puisque toutes les récompenses de l'espace d'états entier ne sont pas nécessairement explorées. Il est néanmoins optimal lorsque les récompenses du problème sont toutes imposées. Il sera donc intéressant de contrôler le mode opportuniste via les récompenses imposées, afin de le rendre le plus optimal possible.

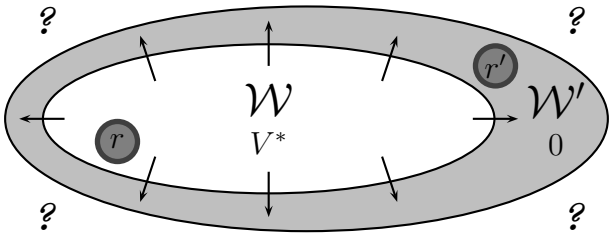


FIG. 9.1 – sfDP découvre les récompenses de manière opportuniste lors de l'expansion de l'espace d'états atteignables

Enfin, avant de détailler notre algorithme, précisons un aspect très important de notre approche. Bien que cela puisse paraître évident, **il est primordial que les sous-buts imposés par la mission correspondent à des récompenses du MDP**. Si cette hypothèse n'est pas vérifiée, la première politique partielle optimisée avec les récompenses du MDP (et non celle de plus sûr chemin stochastique) ne mènera pas aux sous-buts de la mission. Non seulement la politique finale ne permettra pas d'atteindre les sous-buts, mais les états atteignables seront de plus quasiment tous explorés, puisque la condition d'arrêt d'expansion du sous-espace atteignable ne sera jamais vérifiée ($\mathcal{W} \cap \mathcal{B}$ non vide où égal à $\mathcal{B}$).

9.1.3 Fonction sfDP

La fonction sfDP (cf. algorithme 32) commence par calculer l'heuristique de plus sûr chemin stochastique et son espace d'états atteignables. Nous rappelons que cette heuristique est calculée en trois étapes :

1. fonction ExpansionEspaceEtatsAtteignables : obtention d'un premier espace d'états atteignables $\mathcal{W}$, isotrope depuis $\mathcal{I}$ (relaxant les transitions stochastiques)
2. fonction PlusSurCheminStochastique : calcul de la politique de plus sûr chemin stochastique sur $\mathcal{W}$,
3. fonction FiltrageStochastique : filtrage des états de $\mathcal{W}$ qui ont une probabilité négligeable d'atteindre $\mathcal{B}$ par rapport aux états de $\mathcal{I}$, et mise à jour de $\mathcal{W}$.

Ainsi, contrairement à sLAO*, notre heuristique fournit un espace d'états atteignables initial, contenant les sous-buts $\mathcal{B}$.

Ensuite, comme sLAO*, sfDP rentre dans une boucle qui alterne une phase d'expansion de l'espace d'états atteignables, et une phase de programmation dynamique. Néanmoins, comme les heuristiques de sLAO* et sfDP sont radicalement différentes, les fonctions ExpansionEspaceEtatsAtteignables (cf. algorithme 29 page 198) et

Algorithme 32 : Fonction `sfDP`

Données : $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, $\mathcal{I}$, $\mathcal{B}$, γ, ζ, ϵ, *arret_expansion* (booléen : **faible** ou **forte**), *type_expansion* (booléen : **suivre_politique** ou **toutes_actions**)
Résultat : π, V, $\mathcal{W}$
début
  $\mathcal{W} \leftarrow$ `ExpansionEspaceEtatsAtteignables` $(\mathcal{I}, \mathcal{B}, arret\_expansion, \texttt{false})$;
  $(P, \pi) \leftarrow$ `PlusSurCheminStochastique` $(\mathcal{B}, \mathcal{W}, \epsilon)$;
  $\mathcal{W} \leftarrow$ `FiltrageStochastique` $(\mathcal{I}, P, \zeta)$;
  $V \leftarrow$ `ReadZero`;
  $\mathcal{Z} \leftarrow$ `ReadLogicZero`();
  répéter
    si *type_expansion* = **suivre_politique alors**
      $(\mathcal{W}, \mathcal{U}, \mathcal{Z}) \leftarrow$ `ExpansionEspaceEtatsAtteignables` $(\mathcal{I}, \mathcal{B}, arret\_expansion, \textbf{true}, \pi, \mathcal{Z})$;
    $(V, \pi) \leftarrow$ `ProgrammationDynamique` $(\mathcal{W}, \mathcal{V}, \epsilon, type\_expansion)$;
  jusqu'à $\mathcal{U} =$ `ReadLogicZero`();
  retourner $(\pi, V, \mathcal{W})$;
fin

`ProgrammationDynamique` (cf. paragraphe suivant) ont été adaptées aux besoins de `sfDP`.

D'autre part, l'ADD de la fonction de valeur V n'étant pas initialisé (contrairement à `sLAO`*), V est par défaut nulle sur l'espace d'états entier. Ainsi, comme nous l'avons vu (cf. figure 9.1), ceci implique un comportement opportuniste des récompenses du MDP, car la fonction de valeur n'est pas guidée sur la frontière du sous-espace atteignable par les récompenses qui se trouvent en dehors de ce sous-espace. Aussi, nous avons développé deux types d'expansion de l'espace d'états atteignables (cf. figure 9.2) :

- **suivre_politique** : le nouveau sous-espace atteignable $\mathcal{W}'$ est recalculé toujours depuis $\mathcal{I}$ en suivant la politique partielle courante jusqu'à intercepter $\mathcal{B}$ (fonction `ExpansionEspaceEtatsAtteignables`),
- **toutes_actions** : le nouveau sous-espace atteignable $\mathcal{W}'$ est généré en appliquant toutes les actions possibles depuis le précédent sous-espace atteignable $\mathcal{W}'$, indépendamment de la politique partielle courante (réalisé dans la fonction `ProgrammationDynamique`).

Dans le premier cas, proche de `sLAO`* mais avec des conditions d'arrêt d'expansion différentes (cf. algorithme 24 page 186), le sous-espace atteignable est réduit

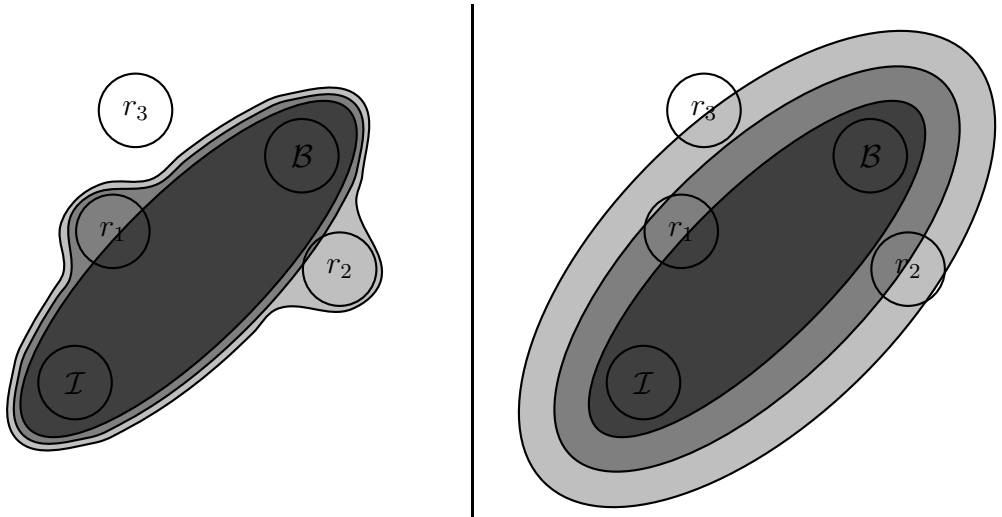


FIG. 9.2 – Deux types possibles d'expansion de l'espace d'états atteignables durant la phase d'optimisation ($\mathcal{W}^0$ foncé, $\mathcal{W}^1$ normal, $\mathcal{W}^2$ clair)

aux récompenses interceptées durant l'expansion. En effet, la politique courante est localement guidée («aspirée») vers les récompenses qui ont une intersection non nulle avec la frontière de l'espace d'états atteignables.

Dans le deuxième cas, le sous-espace atteignable grandit sans cesse, indépendamment des récompenses interceptées durant l'expansion. Comme le montre la figure 9.2, le sous-espace atteignable est potentiellement beaucoup plus grand que dans le premier cas, mais il a plus de chances d'intercepter des récompenses supplémentaires. Ainsi, comme le montrerons nos expérimentations dans la suite, la version **toutes_actions** de `sfDP` est plus lente que la version **suivre_politique**, mais elle est plus opportuniste.

Fonction `ProgrammationDynamique` (`sfDP`)

La fonction `ProgrammationDynamique` de `sfDP` (cf. algorithme 33) est très proche de son homologue de `sLAO*` (cf. algorithme 26 page 188) : la seule différence repose sur l'absence de l'heuristique dans la version de `sfDP`, car elle est incompatible avec la fonction de valeur du MDP.

Dans la version de `sLAO*`, l'heuristique à l'extérieur de l'espace d'états atteignables est sauvegardée dans l'ADD $saveV_{\overline{\mathcal{W}}}$ (cf. algorithme 26 page 188). Cet ADD devient inutile dans la version de `sfDP`, si bien qu'il n'est pas transmis à la fonction `ActualiseFonctionValeur`. En revanche, les fonctions `ImageAction` (cf. algorithme

Algorithme 33 : Fonction `ProgrammationDynamique` (`sfDP`)

Données : $\mathcal{W}$ (BDD), V (ADD), *type_expansion* (booléen : **suivre_politique** ou **toutes_actions**)
Résultat : V, π (liste de BDD)
début
  $saveV \leftarrow V$;
  $\mathcal{W}' \leftarrow$ `ReadLogicZero`();
  pour $a \in \mathcal{A}$ **faire**
    $\mathcal{W}' \leftarrow$ `bddOr`$(\mathcal{W}', $`ImageAction`$(\mathcal{W}, dT^a))$;
  $(R_{\mathcal{W}\cup\mathcal{W}'}, T_{\mathcal{W}\cup\mathcal{W}'}) \leftarrow$ `RestreintTransitions`$(\mathcal{W}, \mathcal{W}')$;
  $primed\_variables \leftarrow$ `addComputeCube` $(\wedge_{i=1}^{n}\{X_i'\})$;
  répéter
    $V' \leftarrow$ `addPermute`$(V, unprimed\_to\_primed)$;
    $V \leftarrow$ `ActualiseFonctionValeur`$(V', \mathcal{W}, R_{\mathcal{W}\cup\mathcal{W}'}, T_{\mathcal{W}\cup\mathcal{W}'}, \textbf{false})$;
    $V_{\mathcal{W}} \leftarrow$ `addApply`$($`addTimes`$, V,$ `BddToAdd`$(\mathcal{W}))$;
    $V'_{\mathcal{W}} \leftarrow$ `addApply`$($`addTimes`$, V',$ `BddToAdd`$(\mathcal{W}))$;
  jusqu'à `EqualSupNorm`$(V_{\mathcal{W}}, V'_{\mathcal{W}}, \epsilon) =$ **true**;
  $(V, \pi) \leftarrow$ `ActualiseFonctionValeur`$(V', \mathcal{W}, R_{\mathcal{W}\cup\mathcal{W}'}, T_{\mathcal{W}\cup\mathcal{W}'}, \textbf{true})$;
  si *type_expansion* = **toutes_actions alors**
    $\mathcal{W} \leftarrow$ `bddOr`$(\mathcal{W}, \mathcal{W}')$;
  retourner (V, π);
fin

25 page 187) et `RestreintTransitions` (cf. algorithme 27 page 189) sont identiques.

Comme dans `sLAO`*, la phase de programmation dynamique de `sfDP` étend l'espace d'états atteignables courant $\mathcal{W}$ en 1 coup en appliquant toutes les actions possibles depuis $\mathcal{W}$. Alors que l'espace d'états obtenu, soit $\mathcal{W}'$, sert dans `sLAO`* à guider la fonction de valeur avec l'heuristique sur la frontière de $\mathcal{W}$ (cf. figure 7.6 page 188), il est utilisé dans `sfDP` pour découvrir opportunément de nouvelles récompenses (cf. figure 9.1).

Fonction `ActualiseFonctionValeur` (`sfDP`)

La fonction `ActualiseFonctionValeur` de `sfDP` (cf. algorithme 34) est quasiment identique à celle définie dans `sLAO`*(cf. algorithme 28 page 190), à ceci près que l'heuristique n'est pas utilisée (représentée par $saveV_{\overline{\mathcal{W}}}$ dans `sLAO`*). En particulier, la version de `sfDP` ne nécessite pas d'opération de masquage de la fonction de valeur V sur $\mathcal{W}$, combinée à l'heuristique $h = saveV_{\overline{\mathcal{W}}}$ sur le complémentaire $\overline{\mathcal{W}}$ (cf. algorithme 28 page 190).

Comme dans `sLAO`*, la politique n'est calculée qu'à la demande afin d'alléger les

Algorithme 34 : Fonction `ActualiseFonctionValeur` (`sfDP`)

Données : V' (ADD avec variables primées), $\mathcal{W}$ (BDD), $R_{\mathcal{W}\cup\mathcal{W}'}$ (ADD), $T_{\mathcal{W}\cup\mathcal{W}'}$ (ADD), *calcule_politique* (booléen)
Résultat : V (ADD avec variables non primées), $[\pi]$ (liste de BDD, retourné si *calcule_politique* = **true**)
début
  pour $a \in \mathcal{A}$ **faire**
    $V^a \leftarrow$ `addApply`$(\texttt{addTimes}, T^a_{\mathcal{W}\cup\mathcal{W}'}, V')$;
    $V^a \leftarrow$ `addExistAbstract`$(V^a, primed\_variables)$;
    $V^a \leftarrow$ `addApply`$(\texttt{addTimes}, \texttt{addConst}(\gamma), V^a)$;
    $V^a \leftarrow$ `addApply`$(\texttt{addPlus}, V^a, R^a_{\mathcal{W}\cup\mathcal{W}'})$;
    si $a = \mathcal{A}.debut()$ **alors**
      $V \leftarrow V^a$;
      si *calcule_politique* = **true alors**
        $\pi[a] \leftarrow \mathcal{W}$;
    sinon
      $V \leftarrow$ `addApply`$(\texttt{addMaximum}, V, V^a)$;
      si *calcule_politique* = **true alors**
        $diff \leftarrow$ `addApply`$(\texttt{addMinus}, V, V^a)$;
        $\pi[a] \leftarrow$ `bddXor`$(\mathcal{W}, \texttt{bddAnd}(\texttt{addBddPattern}(diff), \mathcal{W}))$;
        pour $a' = \mathcal{A}.debut()$ **jusqu'à** $a' = \mathcal{A}.precedent(a)$ **faire**
          $\pi[a'] \leftarrow$ `bddXor`$(\pi[a'], \texttt{bddAnd}(\pi[a'], \pi[a]))$;
  si *calcule_politique* = **true alors**
    retourner (V, π);
  sinon
    retourner V;
fin

calculs. En itération de la valeur, il est en effet inutile de calculer la politique lors de l'amélioration itérative de la fonction de valeur. La politique optimale n'est calculée qu'une seule fois, après que la fonction de valeur ait convergé.

9.2 Replanification en ligne

Lorsque les sous-buts imposés par la mission sont nombreux, l'espace d'états atteignables, ainsi que nous l'expliquons dans la suite, peut être très grand. La phase de programmation dynamique est d'autant plus coûteuse, car sa complexité est polynomiale en le nombre d'états atteignables [14]. En particulier, l'optimisation

du MDP en ligne peut devenir critique, voire impossible, si les sous-buts sont trop nombreux.

Ainsi, une application directe de `sfDP` en ligne peut être désastreuse car, dès qu'une nouvelle récompense est imposée à l'agent, le planificateur optimise de nouveau toute la nouvelle stratégie depuis les états initiaux, indépendamment de la stratégie précédente et des états déjà visités. Ceci est d'autant plus gênant que la complexité de l'optimisation entière de la nouvelle stratégie est polynomiale en la taille du sous-espace atteignable, lui même exponentiel en le nombre de sous-buts.

Ce problème est classique dans le domaine de la replanification en ligne : il est souvent préférable d'opter pour une approche «réactive» qui optimise la mission sur un horizon fini où l'environnement est supposé statique, en replanifiant dès que l'environnement change. Des algorithmes ont été également proposés pour résoudre un problème de planification stochastique en ligne de manière «proactive», en actualisant la fonction de valeur dès que de nouveaux états sont visités par l'agent [97, 98, 95]. Néanmoins, ces derniers ne sont pas réactifs aux changements de l'environnement, comme par exemple l'ajout de nouvelles récompenses dans le modèle, contrairement aux approches réactives.

Ainsi, dans cette section, nous proposons un algorithme intermédiaire entre les approches «réactives» et «proactives» : `IsfDP` («Incremental `sfDP`»). Lorsqu'une nouvelle requête de sous-buts survient, l'agent optimise le MDP en utilisant la stratégie obtenue sur le précédent sous-ensemble atteignable. La nouvelle stratégie n'est donc pas recalculée entièrement depuis le début, mais sur la base des états atteignables courants et des récompenses courantes imposées. Si l'augmentation du sous-espace atteignable est linéaire, le temps d'optimisation de la nouvelle stratégie peut être ainsi quasiment constant et relativement faible au cours du déroulement de la mission, ce que nos résultats expérimentaux confirmeront.

Précisons que notre approche n'est pas «anytime» dans la mesure où nous ne cherchons pas à améliorer une stratégie sous-optimale, optimisée partiellement dès que possible sur un laps de temps limité [99, 34, 100]. Notre algorithme consiste plutôt à actualiser rapidement la stratégie courante en réponse à de nouvelles requêtes de sous-buts, en supposant que le temps est disponible pour mettre à jour la stratégie.

9.2.1 Représentation formelle des buts

Dans le cas général, l'ensemble des sous-buts $\mathcal{B}$ est une conjonction de disjonctions de variables d'état instanciées :

$$\mathcal{B} = \bigwedge_{j=1}^{q} \left(\bigvee_{k=1}^{j_r} \left(X_{i_{j,k}} = x_{i_{j,k}} \right) \right)$$

La conjonction propose un choix de sous-buts à réaliser (chaque instanciation de variables d'état étant vue comme un sous-but), alors que la disjonction impose de réaliser tous les choix de sous-buts : une mission est réussie si au moins un sous-but de chaque choix est réalisé. Par exemple, dans une mission comprenant trois régions géographiques R_1, R_2 et R_3, ainsi que trois objectifs possibles O_1, O_2 et O_3, la contrainte de réalisation des sous-buts peut être "arriver dans R_2 **ou** (au choix) R_3, *et* atteindre O_1 *et* O_2" ($\mathcal{B} = (R_2 \vee R_3) \wedge O_1 \wedge O_2$).

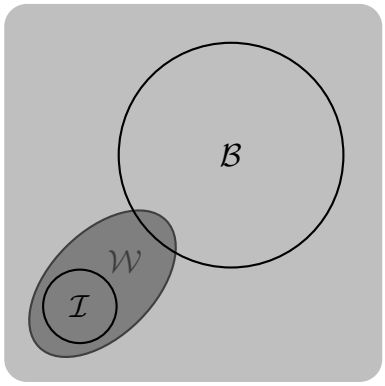
Une disjonction de variables d'état instanciées correspond à un sous-ensemble d'états de l'espace entier, puisque ce dernier est égal à la réunion de toutes les variables d'état sur toutes leurs valeurs possibles. Ainsi, une conjonction de choix de variables d'état instanciées correspond à un sous-ensemble d'états de l'espace entier, en tant qu'intersection de sous-ensembles d'état. Aussi, plus le nombre de choix de sous-buts est grand, plus le sous-ensemble de ces états-buts est petit.

Par conséquent, la recherche des choix de sous-buts dans l'espace entier, par expansion du sous-espace atteignable, est d'autant plus difficile que les choix de sous-buts imposés sont nombreux (cf. figure 9.3), à moins que le sous-ensemble de buts soit proche des états initiaux en termes de nombre d'actions à réaliser pour y parvenir. Il en résulte un sous-espace d'états atteignables grand, dans lequel l'optimisation partielle du MDP sera difficile. Notons que les choix de sous-buts peuvent être nombreux sans que les choix soient variés : il suffit de considérer des choix réduits à une seule instanciation de variables d'état.

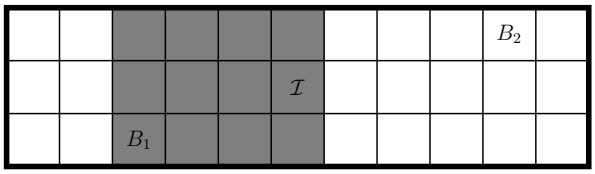
Le sous-ensemble atteignable risque donc d'être grand, ce qui peut limiter l'application de `sfDP` en ligne. Ainsi, nous proposons dans le paragraphe suivant un algorithme incrémental appelé `IsfDP` («Incremental `sfDP`»), qui optimise le MDP sur une liste partielle et temporaire de choix de sous-buts à réaliser, en augmentant petit à petit le nombre de contraintes de sous-buts. Pour plus d'efficacité, la fonction de valeur optimisée avec la liste courante de sous-buts est réutilisée à l'itération suivante (incrément de la liste de sous-buts imposés). Dès que la nouvelle stratégie est obtenue, l'agent change de stratégie même s'il n'a pas atteint tous les sous-buts de la précédente stratégie : en effet, la nouvelle stratégie prend également en compte les sous-buts imposés à la précédente stratégie (cf. figure 9.4).

A chaque itération, la stratégie partielle est optimisée avec `sfDP` en mode opportuniste. Si, à un moment de l'exécution, toutes les récompenses sont imposées à l'agent, `sfDP` est alors appelé en mode optimal, si bien que la stratégie recalculée en ligne devient optimale. Par conséquent, `IsfDP` possède le même comportement opportuniste que `sfDP`, dont le cas limite (toutes les récompenses imposées) est optimal.

Par ailleurs, les buts peuvent se modéliser, de manière duale, sous forme d'un choix de sous-buts imposés (disjonction de conjonctions). Dans cette représentation, le sous-espace atteignable peut être petit si un des choix impose un faible nombre

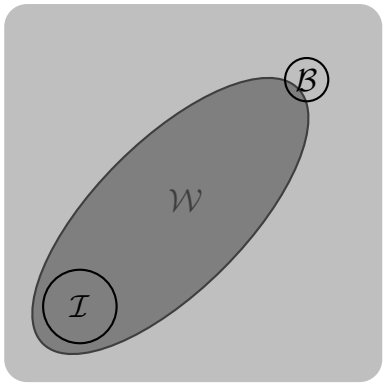


espace d'états entiers

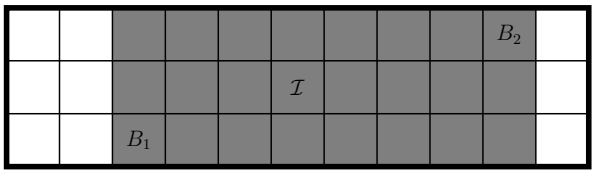


sous-espace de navigation

$\mathcal{B} = (B_1 \textbf{ atteint})$



espace d'états entiers



sous-espace de navigation

$\mathcal{B} = (B_1 \textbf{ atteint}) \wedge (B_2 \textbf{ atteint})$

FIG. 9.3 – Taille de l'espace atteignable et nombre de sous-buts imposés (exemple d'un espace d'états globalement factorisé, avec une composante de navigation énumérée, et des choix réduits chacun à une seule variable d'état instanciée)

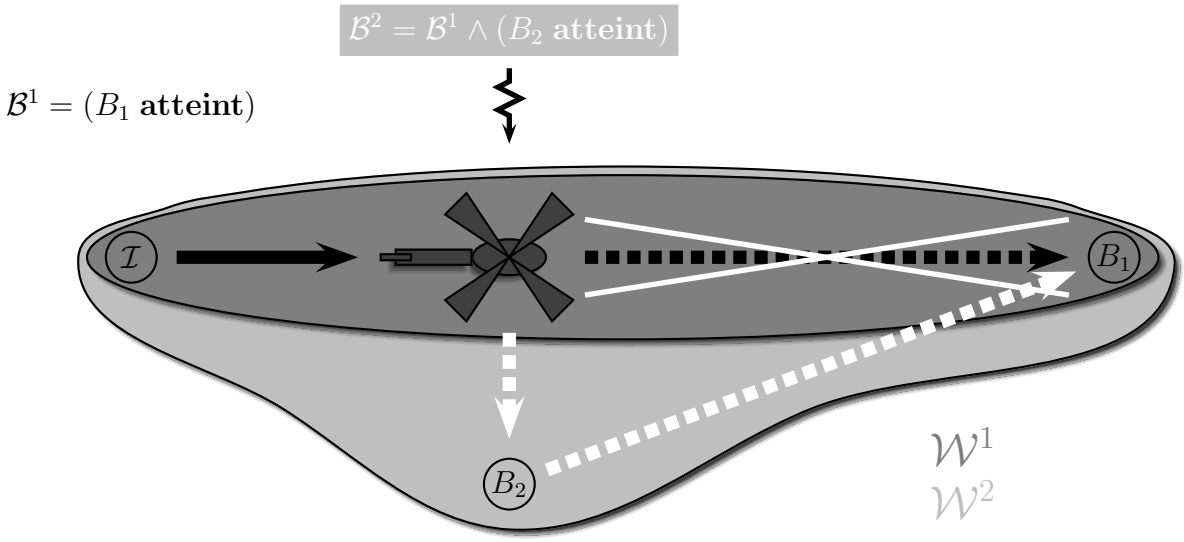


FIG. 9.4 – `IsfDP` replanifie la stratégie lorsqu'une nouvelle requête de sous-buts survient, en utilisant le sous-espace atteignable et la stratégie optimisée obtenus avec les précédentes récompenses imposées.

de sous-buts à réaliser *et* que la condition d'arrêt de l'algorithme d'expansion des états atteignables est faible (cf. algorithme 29 page 198). En effet, dans ce cas, l'expansion du sous-espace atteignable s'arrête lorsqu'un des choix de sous-buts est atteint (intersection avec $\mathcal{B}$ non vide), en particulier celui qui correspond à un faible nombre de sous-buts à réaliser.

9.2.2 `IsfDP` : une version incrémentale de `sfDP`

Nous supposons que $\mathcal{B}$ est une conjonction de k choix imposés de sous-buts :

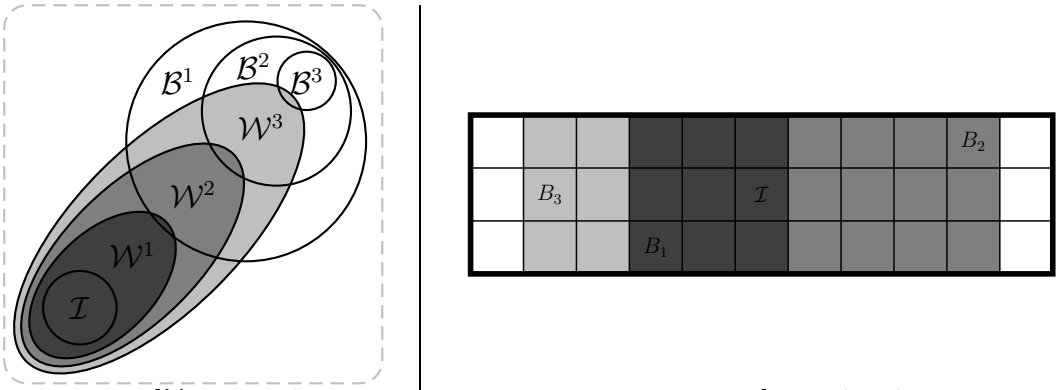
$$\mathcal{B} = \bigwedge_{j=1}^{q} \bigvee_{k=1}^{j_r} \left(X_{i_{j,k}} = x_{i_{j,k}}\right)$$

L'ensemble $\mathcal{B}$ peut s'écrire de manière récursive :

- $\mathcal{B}^0 = \mathcal{V}$
- $\forall\ 1 \leqslant j \leqslant q\ ,\ \mathcal{B}^j = \mathcal{B}^{j-1} \bigwedge \left(\bigvee_{k=1}^{j_r} \left(X_{i_{j,k}} = x_{i_{j,k}}\right) \right)$
- $\mathcal{B} = \mathcal{B}^q$

L'algorithme `IsfDP` résout itérativement le MDP à l'aide de `sfDP` sur chaque ensemble *incrémental* de buts $\mathcal{B}^k$, mais *décroissant* (cf. figure 9.5) :

$$\forall\ 1 \leqslant j \leqslant q\ ,\ \mathcal{B}^j \subset \mathcal{B}^{j-1}$$



espace d'états entiers | sous-espace de navigation

FIG. 9.5 – `IsfDP` optimise le MDP avec une liste incrémentale mais décroissante de sous-buts à réaliser, et donc sur un ensemble croissant de sous-ensembles atteignables. (exemple d'un espace d'états globalement factorisé, avec une composante de navigation énumérée : $\mathcal{B}^1 = (B_1$ **atteint**$)$, $\mathcal{B}^2 = \mathcal{B}^1 \wedge (B_2$ **atteint**$)$, $\mathcal{B}^3 = \mathcal{B}^2 \wedge (B_3$ **atteint**$) = \mathcal{B}$)

Ainsi, comme nous l'avons expliqué dans le paragraphe précédent, `sfDP` optimise plus facilement le MDP avec $\mathcal{B}^{j-1}$ comme objectif à réaliser, plutôt qu'avec $\mathcal{B}^j$ (puisque $\mathcal{W}^j$ est plus grand que $\mathcal{W}^{j-1}$). Par conséquent, l'algorithme `IsfDP` (cf. algorithme 35) appelle `sfDP` itérativement sur $\mathcal{B}^j$, en utilisant la solution obtenue à l'itération précédente sur $\mathcal{B}^{j-1}$.

Les q choix de sous-buts $(\mathcal{B}^j)_{1 \leqslant j \leqslant q}$ correspondent chacun à l'union de variables d'état instanciées (`bddOr`), chaque instanciation étant représentée sous forme de BDD :

$$\forall\, 1 \leqslant j \leqslant q\ ,\ \mathcal{B}^j = \bigvee_{1 \leqslant k \leqslant j_r} \underbrace{\left(X_{i_{j,k}} = x_{i_{j,k}}\right)}_{B_{j,k}}$$

Les BDD $\left((B_{j,k})_{1 \leqslant k \leqslant j_r}\right)_{1 \leqslant j \leqslant q}$ peuvent être initialisés à l'aide de la fonction `bddIthVar`(i) de la librairie de diagrammes de décision (voir par exemple [70]), qui renvoie le BDD $\chi_{(X_i = \mathbf{true})}$. Le sous-ensemble de buts courant $(\mathcal{B})$ est initialisé sur l'ensemble d'états entier (`ReadOne`), puis les choix de sous-buts lui sont ajoutés incrémentalement par intersection (`bddAnd`).

Afin de faciliter la recherche du sous-ensemble de buts courant dans le calcul d'heuristique de `sfDP`, les états initiaux de `sfDP` (deuxième argument) sont initialisés à $\mathcal{W}$, qui est le précédent sous-ensemble d'états atteignables. Ainsi, à chaque appel j de `sfDP`, les nouveaux états atteignables ne sont pas recalculés depuis $\mathcal{I}$, mais depuis $\mathcal{W}^{j-1}$ jusqu'à $\mathcal{B}^j$.

Algorithme 35 : Fonction `IsfDP`

Données : $\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, $\mathcal{I}$, $\left((B_{j,k})_{1 \leqslant k \leqslant j_r}\right)_{1 \leqslant j \leqslant q}$ (liste de BDD), γ, ζ, ϵ, *arret_expansion*, *type_expansion*
Résultat : π, V, $\mathcal{W}$
début
  $\mathcal{W} \leftarrow \mathcal{I}$;
  $\mathcal{B} \leftarrow$ `ReadOne`();
  $V \leftarrow$ `ReadZero`();
  pour $1 \leqslant j \leqslant q$ **faire**
    $\mathcal{B}^j \leftarrow$ `ReadLogicZero`();
    pour $1 \leqslant k \leqslant j_r$ **faire**
      $\mathcal{B}^j \leftarrow$ `bddOr` $(\mathcal{B}^j, B_{j,k})$;
    $\mathcal{B} \leftarrow$ `bddAnd`$(\mathcal{B}, \mathcal{B}^j)$;
    $(\pi, V, \mathcal{W}) \leftarrow$ `sfDP`$(\langle \mathcal{V}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle, \mathcal{W}, \mathcal{B}, V, \gamma, \zeta, \epsilon,$ *arret_expansion*, *type_expansion*$)$;
  retourner $(\pi, V, \mathcal{W})$;
fin

Enfin, la fonction de valeur a été rajoutée aux arguments de `sfDP`, afin que l'optimisation de la stratégie courante ne soit pas recalculée entièrement, et qu'elle utilise celle calculée sur le précédent sous-ensemble atteignable. Aussi, dans `sfDP` (cf. algorithme 32 page 222), la fonction de valeur n'est plus ré-initialisée à 0 (la ligne "$V \leftarrow$ `ReadZero`() ;" est supprimée).[1]

9.2.3 Bilan des différentes versions de `sfDP`

Le tableau 9.1 présente les différentes versions de `sfDP` que nous avons implémenté, suivant les options de condition d'arrêt et de type de l'expansion de l'espace d'états atteignables. Les 8 versions obtenues utilisent toutes la même base algorithmique, qui peut être considéré comme une *classe d'algorithmes de type* `sfDP`.[2] Les notations de ces différentes versions sont utilisées dans la section suivante, afin de comparer les performances des algorithmes.

[1] En itération de la politique, π serait transmis à `sfDP` à la place de V.

[2] En considérant les versions «itération de la politique» de ces algorithmes, que nous avons également implémenté, la classe d'algorithmes `sfDP` contient 16 versions différentes.

Version	Condition d'arrêt	Type d'expansion	Buts incrémentaux
PsfDP	faible	suivre_politique	NON
AsfDP	faible	toutes_actions	NON
PSFDP	forte	suivre_politique	NON
ASFDP	forte	toutes_actions	NON
IPsfDP	faible	suivre_politique	OUI
IAsfDP	faible	toutes_action	OUI
IPSFDP	forte	suivre_politique	OUI
IASFDP	forte	toutes_actions	OUI

TAB. 9.1 – Bilan des différentes versions de sfDP

9.3 Expérimentations

9.3.1 Compétition Internationale de Planification Stochastique (ICAPS 2004)

Présentation de la compétition

Nous avons participé à la Première Compétition Internationale de Planification Stochastique [101], qui a été organisée à l'occasion de la Conférence Internationale de Planification et Ordonnancement Automatique (ICAPS) au Canada en 2004. Cette compétition s'inscrit dans une lignée de compétitions internationales de planification déterministes, dont la première eu lieu en 1998 [102]. Ainsi, depuis 2004 et tous les deux ans, une nouvelle partie probabiliste se déroule en parallèle de la partie déterministe.

En tant que participant, nous avons développé une version initiale de sfDP pour les besoins de la compétition. Les problèmes de planification stochastique de la compétition sont définis dans un langage propre appelé PPDDL [103], qui est une extension probabiliste du langage de planification PDDL [104], lui-même utilisé dans la partie déterministe de la compétition. Ce langage est divisé en deux sections (cf. annexe D page 283) :

- *domaine* : définit les états, les actions et les transitions stochastiques
- *problème* : donne les états initiaux possibles, et les états buts imposés

Les états du domaine sont des instanciations d'objets de l'environnement, qui ont chacun plusieurs caractéristiques possibles. Aussi, ces caractéristiques permettent de définir l'espace d'états (d'objets instanciés) sous une forme factorisée avec des variables d'état binaires [101]. Les actions et leurs transitions sont définies sous une forme fonctionnelle implicite, où les effets probabilistes sont donnés en fonction d'objets non instanciés. Ce formalisme est différent des MDP factorisés pour lesquels les actions, aussi compactes soient-elles, sont des représentations structurées de matrices

de transitions instanciées. Les organisateurs ont donc fourni un traducteur de PPDDL vers DBN (réseau bayésien générique), pour les participants dont le planificateur était de type MDP [101].

Le problème, qui définit en quelques sortes la mission, est indépendant du domaine. Ainsi, plusieurs problèmes peuvent être définis pour un même domaine. Inspiré de la partie déterministe de la compétition, le problème comprend des **états initiaux possibles équiprobables**, et des **états buts qui ne correspondent pas nécessairement à toutes les récompenses du domaine**. Ce dernier point désavantage les planificateurs MDP, si bien que deux évaluations différentes basées sur les buts atteints et les récompenses accumulées ont été effectuées. C'est aussi la raison pour laquelle nous avons développé un algorithme d'optimisation heuristique focalisé sur les buts du problème.

Enfin, l'optimisation du problème s'effectue à distance, comme le montre la figure 9.6. L'évaluation de chaque essai — 30 en tout — comprend trois phases, qui ne doivent pas excéder 15 minutes au total (indépendamment du problème) :

1. le serveur des organisateurs envoie par sockets (format XML) un état initial tiré au hasard dans la liste des états initiaux possibles ;
2. le client (compétiteur) optimise la stratégie connaissant l'état initial et les buts du problème
3. simulation et évaluation de la stratégie : le serveur et le client dialoguent par sockets (format XML) en s'échangeant successivement l'action à réaliser dans l'état courant (client) et l'état suivant tiré au hasard dans la liste des successeurs stochastiques de l'état courant pour cette action (serveur).

La phase d'optimisation de la stratégie par le client peut, bien entendu, être effectuée au cours de la simulation.

Inconvénients du protocole de communication, et du traducteur PPDDL vers DBN

Le protocole de communication implique que chaque participant résout le problème sur un ordinateur de son choix, qui n'est pas connu des évaluateurs. Ceci est particulièrement regrettable dans la mesure où le temps de calcul de la stratégie est pris en compte dans l'évaluation de certains problèmes (ceux basés sur les buts atteints). De même, le temps de communication entre le serveur et les clients n'est malencontreusement pas soustrait du temps d'évaluation total, ce qui désavantage les participants éloignés du serveur (hébergé aux États-Unis), d'autant plus que les communications sont très nombreuses durant la simulation de la stratégie.

Par ailleurs, le langage PPDDL est très pratique, car le formalisme implicite et fonctionnel des actions permet de modéliser facilement des domaines complexes de très grandes tailles [103]. Néanmoins, si ce formalisme est particulièrement bien

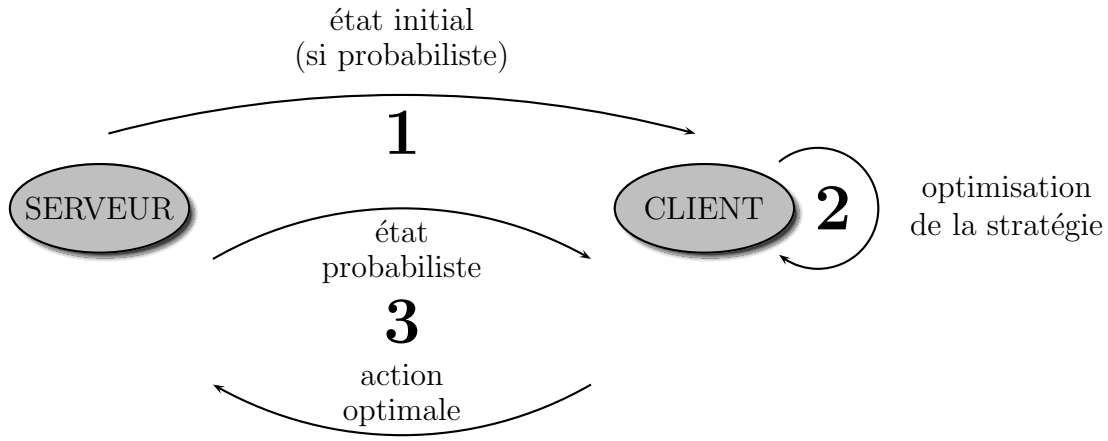


FIG. 9.6 – Première Compétition Internationale de Planification : protocole de communication entre le serveur distant et les participants ($\{1 + 2 + 3\} = 15\ mn$)

adapté aux planificateurs en-ligne (qui n'instancient les transitions qu'au besoin), son avantage est totalement perdu pour les planificateurs hors-ligne : ces derniers nécessitent d'instancier toutes les transitions, ce qui peut devenir problématique pour des domaines très complexes. Dans notre cas, la traduction des domaines en MDP factorisés, à l'aide du logiciel fourni par les organisateurs, durait plus de 15 minutes, si bien que nous n'avions même pas le temps de commencer l'optimisation du problème.[3]

En outre, les stratégies obtenues pour chaque problème sont évaluées par simulation. D'après la loi des grands nombres, de nombreux essais doivent être effectués afin d'évaluer correctement les gains cumulés depuis les états initiaux. Or, seuls 30 essais par problème ont été réalisés ce qui, selon nous, est insuffisant. Aussi, force est de constater que les planificateurs MDP ne sont pas adaptés au langage `PPDDL` et l'évaluation des stratégies par simulation. Ainsi, suite à de nombreuses critiques durant la compétition, une évaluation a été menée séparément, qui ne prenait pas en compte le temps de calcul et qui se basait uniquement sur les gains cumulés lors des essais.

Planificateurs en compétition

Vingt planificateurs étaient inscrits à la compétition. Lors de l'évaluation, plus que 7 planificateurs participaient, dont leurs identifiants sont :

[3]Nous avons également constaté que la somme des probabilités des états successeurs pour une action donnée, après traduction en MDP factorisé, était dans certains cas strictement inférieure à 1. Dans ces conditions, il est impossible d'obtenir une stratégie optimale.

- C [105] : recherche symbolique heuristique (`sLAO*`)
- E [106] : itération de la valeur du premier ordre avec des calculs propositionnels, spécifique au domaine (`FCPlanner`)
- G [107] (`NMRDPP`) :
  - G1 : exploitation de récompenses non markoviennes
  - G2 : `NMRDPP` augmenté de contrôle de connaissance
- J [108] :
  - J1 : politique écrite par intervention humaine, dans un langage de classification de politiques
  - J2 : itération de la politique hors-ligne, en acquérant automatiquement une politique spécifique au domaine
  - J3 : replanification déterministe en utilisant FF (`FFrePlan`)
- P [109] : `LRTDP` avec des bornes inférieures extraites de la description du problème (`mGPT`)
- Q [110] : planificateur de type POP sans observation (`Probapop`)
- R [111] : première version de `sfDP` (notre planificateur)

Ces planificateurs sont détaillés dans les références données ci-dessus. Notre planificateur (R) est une version de `sfDP` qui dont l'expansion des états atteignables est intermédiaire entre **toutes_actions** et **suivre_politique** : les nouveaux états atteignables sont calculés en appliquant la politique courante en un coup depuis le précédent sous-espace atteignable. Ce type d'expansion s'est avéré inefficace car il augmente sans cesse le sous-espace atteignable sans avoir vraiment l'opportunité de rencontrer de nouvelles récompenses (car toutes les actions ne sont pas appliquées). Quitte à suivre la politique courante, autant générer de nouveau le sous-espace atteignable depuis les états initiaux, ce qui permet de mieux cibler les buts. Ainsi, suite à la compétition de planification, nous avons défini les deux types actuels et opposés d'expansion : **toutes_actions** et **suivre_politique**.

Enfin, précisons que notre première version de `sfDP` (celle utilisées dans la compétition) avait un bogue au niveau du filtrage stochastique des états atteignables, si bien que le sous-espace atteignable n'était pas réduit après le calcul de la politique de plus sûr chemin stochastique. Cependant, les résultats officiels, que nous présentons dans le paragraphe suivant, montrent que notre planificateur a obtenu de bons résultats dans la catégorie «cumul des récompenses».

Résultats officiels

Les planificateurs ont été évalués suivant 5 catégories différentes, dont 2 seulement prenaient en compte des récompenses [101] :

- tous les domaines sont évalués ; la valeur d'un planificateur est une pondération entre le temps d'évaluation total (calcul de la stratégie et simulation), et le nombre de buts atteints (chacun but est récompensé de 500)

– une partie des domaines est évaluée ; la valeur d'un planificateur est la récompense accumulée durant les simulations sans prendre en compte le temps d'évaluation, et les buts n'ont pas de récompenses spécifiques.

Ainsi, seule la deuxième catégorie d'évaluation évalue les planificateurs MDP. En effet, la première catégorie récompense chaque but atteint de 500, mais cette récompense artificielle n'est pas dans le domaine si bien que les planificateurs MDP ne peuvent pas la prendre en compte dans l'optimisation de la stratégie. Précisons que chaque problème n'a été évalué qu'une seule fois pour toutes les catégories et que les critères d'évaluation n'étaient pas connus lors des essais. Il était donc impossible de prendre en compte les récompenses artificielles des buts dans la première catégorie.

La figure 9.7 présente les résultats obtenus pour la première catégorie d'évaluation. Deux planificateurs se démarquent, tous les autres étant à peu près équivalents. Le meilleur (J3) est un planificateur déterministe, et le deuxième (P) est un planificateur en-ligne. Ainsi, ces résultats montrent que la première catégorie d'évaluation était favorable aux planificateurs qui optimisent la stratégie en-ligne, puisqu'ils pouvaient intégrer à leur stratégie la récompense de 500 sur les buts rencontrés, contrairement aux planificateurs hors-ligne. De plus, cette catégorie d'évaluation ne nous semble pas révélatrice des performances comparées des planificateurs, puisqu'elle prend en compte le temps de calcul et de communication, qui est propre à chaque participant, indépendamment du planificateur.

La figure 9.8 présente les résultats obtenus dans la deuxième catégorie d'évaluation. Il apparaît que le premier planificateur est `sLAO*` (C), basé sur les MDP. Notre planificateur (R) est en troisième position juste derrière G1, qui nous devance légèrement sur le dernier problème. Ainsi, notre planificateur est bien placé au regard de la maximisation des récompenses cumulées, où se situent en particulier les planificateurs MDP. Nous tenons à préciser que notre planificateur actuel est bien plus mûr que lors de la compétition :

– la phase de filtrage stochastique a été déboguée si bien que le sous-espace atteignable est dès lors nettement réduit
– l'expansion du sous-espace atteignable est actuellement soit **toutes_actions** soit **suivre_politique**, dont nous verrons dans la section suivante qu'elle est beaucoup plus performante que **toutes_actions** (proche dans son principe de celle utilisée lors de la compétition)

Ainsi, au vu de ces résultats encourageants, il serait intéressant de participer à la prochaine compétition qui aura lieu en 2006, avec le planificateur `sfDP` actuel. La section suivante présente des tests réalisés sur des missions RESSAC avec le planificateur `sfDP` actuel, qui correspond exactement au formalisme développé dans cette thèse.

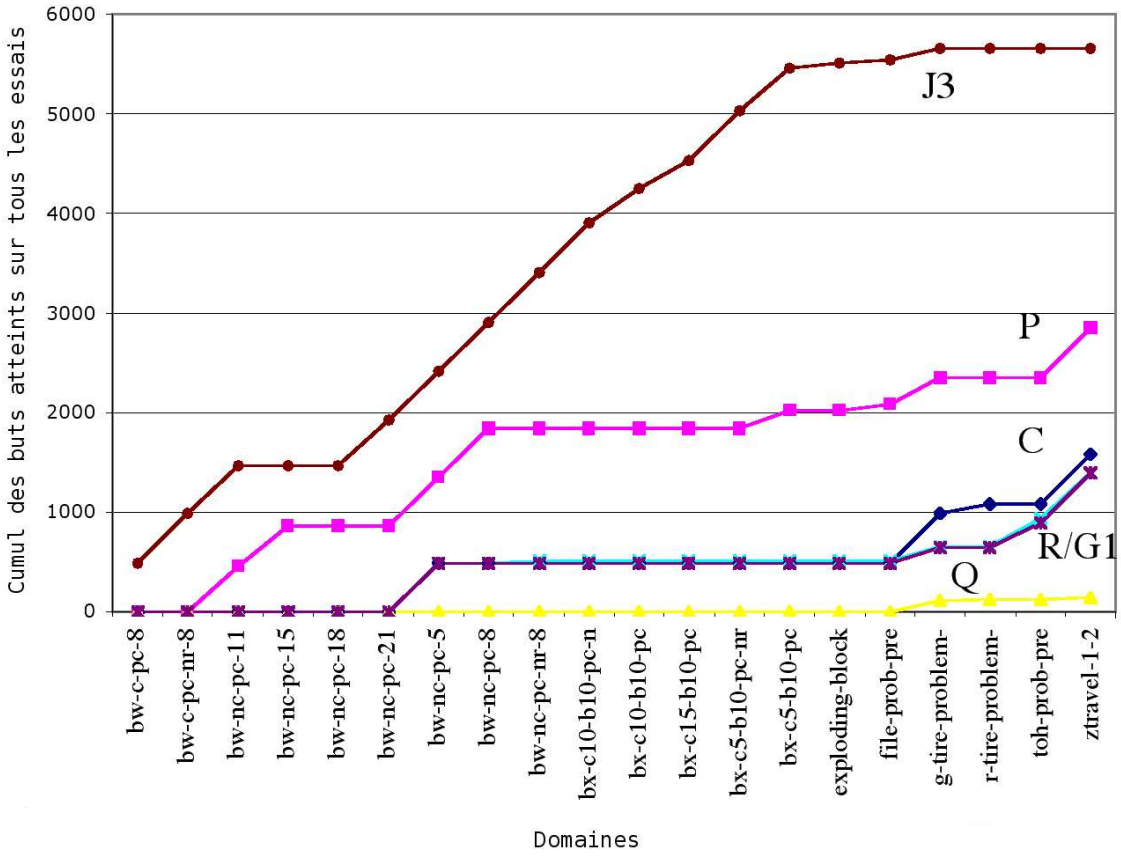


FIG. 9.7 – Première Compétition Internationale de Planification : récompense des buts atteints sur l'ensemble des domaines (pondérés avec le temps d'évaluation total)

9.3.2 Missions ReSSAC

Nous avons testé les différentes versions de `sfDP` (cf. tableau 9.1) sur des missions de type ReSSAC. Afin d'évaluer les performances de ces algorithmes, l'ordre de grandeur de la complexité des missions varie de 10^5 états (facile) à 10^{12} états (très difficile). Au vu de la dimension du plus grand problème testé, nous avons généré automatiquement des problèmes ReSSAC, basés sur des «grilles» de navigation.

Chaque problème testé a trois types de variables :

- variable de navigation, partitionnée en régions ; la topologie du sous-espace de navigation est concentrique, calquée sur la «grille concentrique» de la figure 6.24 (page 172),
- variables de sous-buts imposés par la mission : 1 sous-but par région, et donc autant de variables sous-buts que de régions,
- variable binaire d'autonomie.

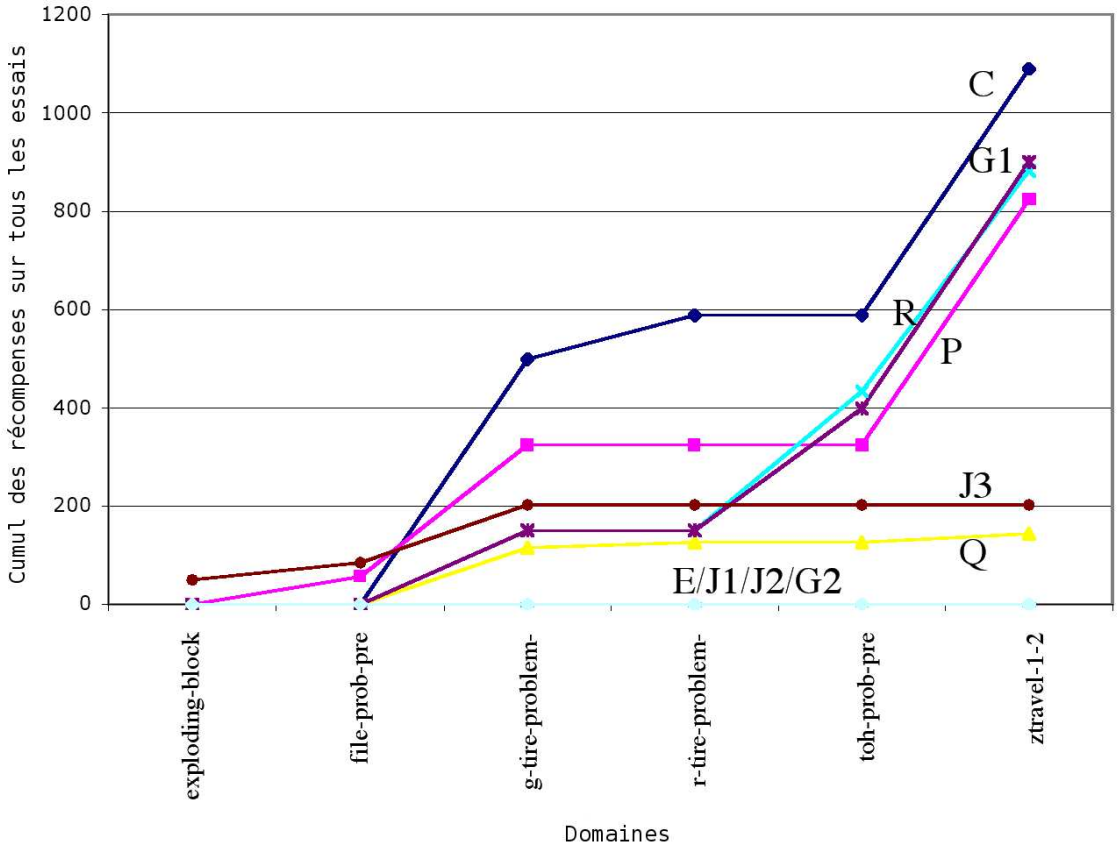


FIG. 9.8 – Première Compétition Internationale de Planification : récompenses obtenues sur les domaines autres que «blockworlds» et «boxworlds»

La taille de l'espace d'états énumérés est : $2^{n_r+1}n_e$, où n_e est le nombre d'états de navigation et n_r est le nombre de régions.

Nous avons testé 6 problèmes «concentriques» différents, de topologies identiques, mais qui diffèrent par le nombre d'états de navigation par région carrée (9 ou 81) et le nombre de couches concentriques (de 1 à 3). Le noyau central est constitué d'une seule région carrée et chaque couche renferme 8 régions dont 4 au moins sont carrées (cf. figure 6.24 page 172). Le problème le plus simple a donc une taille de $2^{10}\times 9\times 9 = 82944$, alors que celle du plus difficile est de $2^{26}\times 7^2\times 81 = 266355081216$.

D'autre part, l'ensemble des états initiaux est une région de départ de la couche concentrique périphérique *ou* la région centrale. Les sous-buts imposés par la mission sont également des régions de la couche périphérique. Une mission consiste donc à atteindre les sous-buts de certaines régions périphériques imposées, en partant d'une région périphérique donnée. Il n'est pas interdit de réaliser d'autres sous-buts, qui pourront être atteints par `sfDP` s'il en a l'opportunité (durant la phase d'expansion

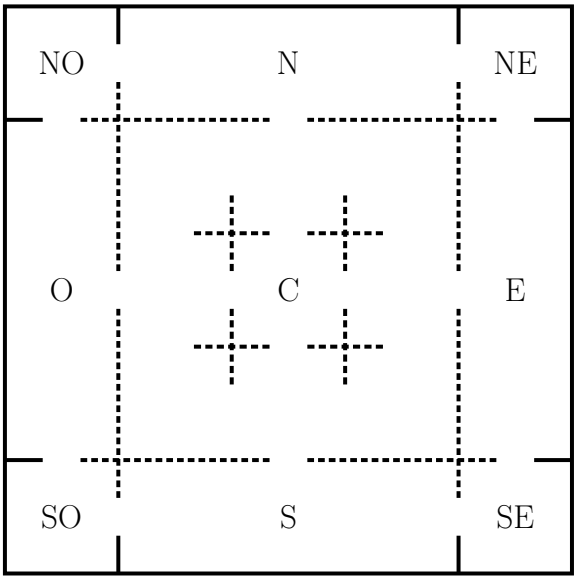


FIG. 9.9 – Positions cardinales des régions de la couche périphérique des missions «concentriques» (cf. figure 6.24 page 172)

du sous-espace atteignable).

Enfin, les régions de la couche périphérique sont identifiées par leur position cardinale, comme le montre la figure 9.9. Les paramètres d'optimisation sont : $\gamma = 0,9$, $\epsilon = 0,1$ et $\zeta = 1$. Le coefficient de filtrage est relativement grand car, ainsi que nous l'avons expliqué dans la section 8.4 (page 214), un filtrage trop large risque de couper de nombreuses trajectoires optimales. Néanmoins, comme nous allons l'expérimenter, la perte d'optimalité est au pire de 3% sur l'ensemble des problèmes que nous avons testés.

Condition d'arrêt forte, différents types d'expansion

Nous avons d'abord testé l'influence des différents type d'expansion (**suivre_politique** ou **toutes_actions**), pour la condition d'arrêt **forte** de l'expansion du sous-espace atteignable ($\mathcal{B} \subset \mathcal{W}$). Les deux algorithmes comparés sont donc respectivement `PSFDP` et `ASFDP`.

Dans un premier temps, nous avons testé le mode opportuniste de `PSFDP` et `ASFDP` pour des problèmes de tailles différentes. Les missions consistent toutes à rejoindre la région NE depuis différentes régions de départ, et à atteindre la récompense de la région NE (cf. figure 9.9). Le but de l'optimisation du MDP est donc de récolter la récompense NE, en autorisant d'atteindre d'autres récompenses si possible.

La figure 9.10 montre le temps de calcul de la stratégie partielle (définie sur le sous-espace atteignable final) pour des missions de différentes tailles. La complexité

des missions augmente, dans l'ordre, avec le nombre de régions (nombre total de variables sous-buts), le nombre d'états par région carrée, et la longueur du trajet (éloignement de la région initiale et de la région but). Nous remarquons les points suivants :

- l'algorithme optimal et complet (SPUDD) ne peut résoudre que la plus petite mission en 10 fois plus de temps que ASFDP et 100 fois plus de temps que PSFDP ;
- ASFDP ne peut résoudre que des missions petites ou moyennes (moins de 17 régions et 9 états par région carrée), sur des petits trajets ($\mathcal{I}$ et $\mathcal{B}$ proches), et en 10 fois plus de temps que PSFDP ;
- PSFDP peut résoudre de grands problèmes (jusqu'à 17 régions et 81 états par région), beaucoup plus rapidement que SPUDD et ASFDP ;
- pour une mission de taille donnée, le temps de calcul de PSFDP et ASFDP dépend du trajet parcouru.

Ces résultats montrent donc que les algorithmes PSFDP et ASFDP n'explorent pas tout l'espace d'états, puisque le temps de calcul dépend de la distance entre la région initiale et la région but. De plus, PSFDP est plus performant que ASFDP.

Intuitivement, d'après la figure 9.2 (page 223), ASFDP explore plus d'états que PSFDP car son sous-espace atteignable augmente indépendamment de la politique courante, c'est-à-dire des sous-buts visés. Autrement dit, ASFDP est moins focalisé sur les sous-buts imposés par la mission que PSFDP. Ceci est vérifié expérimentalement par la figure 9.11, qui représente à la fois le pourcentage du sous-espace atteignable par rapport à l'espace total, et l'évolution relative de la taille du sous-espace atteignable entre la fin du calcul de l'heuristique et la fin du calcul de la stratégie partielle :

- la taille du sous-espace atteignable développé par ASFDP est du même ordre de grandeur que celle de l'espace entier ;
- la taille du sous-espace atteignable de ASFDP a augmenté au moins de moitié, voire de sa totalité, durant l'optimisation de la stratégie ;
- la taille du sous-espace atteignable développé par PSFDP est négligeable devant celle de l'espace entier (quasiment nulle) ;
- la taille du sous-espace atteignable de PSFDP a diminué quasiment de sa totalité durant l'optimisation de la stratégie.

Ainsi, comme prévu, PSFDP focalise le sous-espace atteignable sur les sous-buts, ce qui l'amène à réduire la taille de ce sous-espace au fur et à mesure que la politique se concentre sur les sous-buts (durant l'optimisation de la stratégie). Au contraire, ASFDP se disperse : le sous-espace atteignable calculé par l'heuristique augmente ensuite durant l'optimisation de la stratégie, ce qui signifie qu'il est de moins en moins focalisé sur les sous-buts. De plus, PSFDP exploite la politique de l'heuristique de plus sûr chemin stochastique, alors que ASFDP n'utilise que le sous-espace attei-

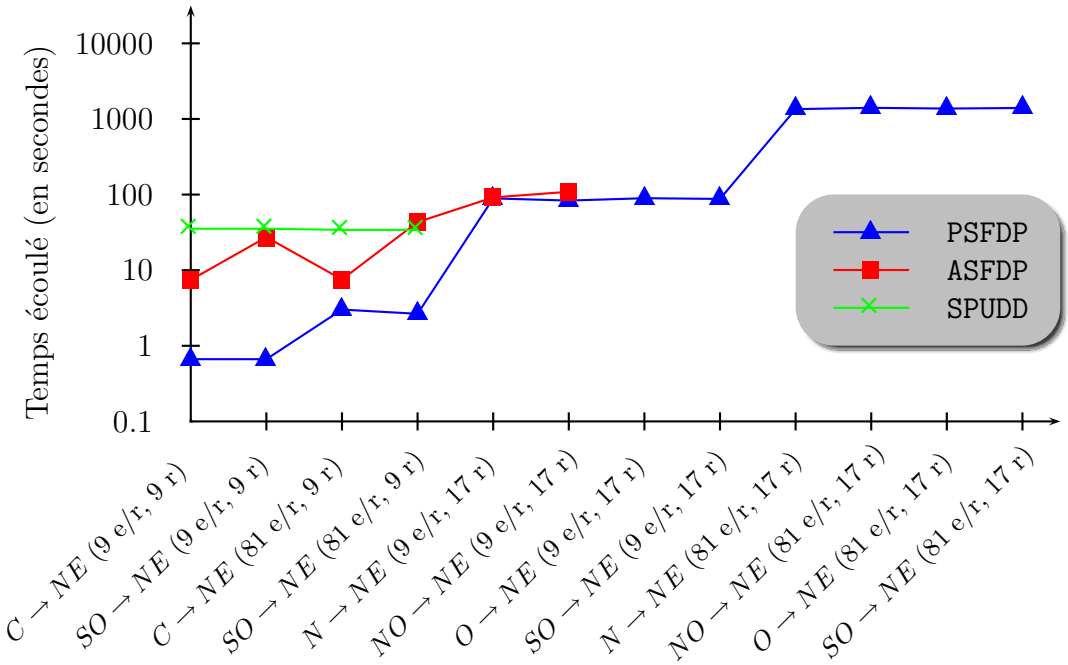
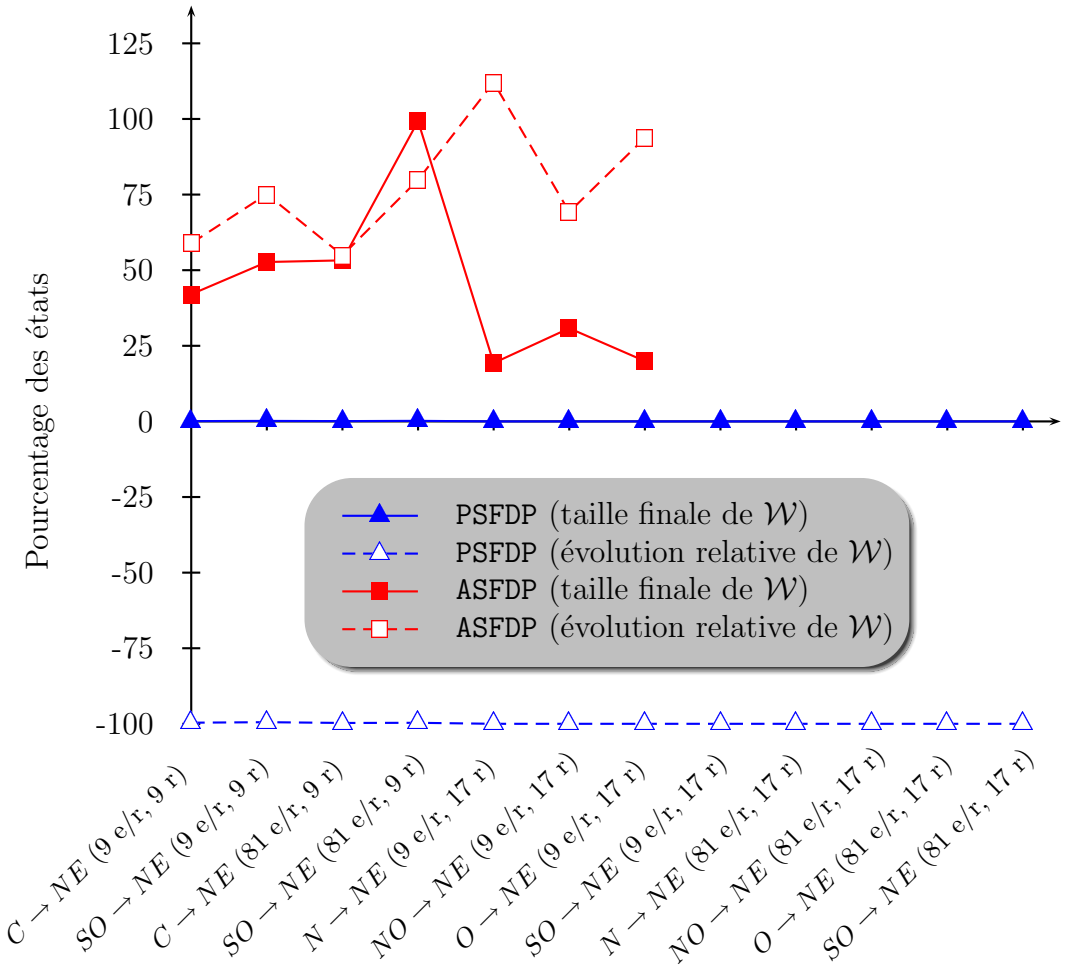


FIG. 9.10 – Mode opportuniste : comparaison du temps de calcul entre `PSFDP`, `ASFDP` et `SPUDD` (en secondes avec un processeur P4 de 2,8 GHz), pour des grilles concentriques de différentes tailles et des trajets différents, en imposant le sous-buts NE (cf. figure 9.9)

gnable calculé par l'heuristique, puisque ce dernier n'utilise pas la politique courante pour étendre le sous-espace atteignable courant. Par conséquent, `PSFDP` **bénéficie de l'information initiale supplémentaire sur la direction la plus probable vers les sous-buts, ce qui guide mieux l'expansion du sous-espace atteignable.**

Néanmoins, `PSFDP` a moins de chances que `ASFDP` de découvrir des récompenses externes au sous-espace atteignable, puisqu'il explore beaucoup moins d'états. La figure 9.12 montre la valeur optimisée de la région initiale obtenue avec une stratégie visant à aller de la région SO à la région NE d'une mission concentrique à 9 régions, pour un nombre croissant de sous-buts imposés. `SPUDD` est optimal si bien que sa valeur indique la récompense obtenue si *tous* les buts possibles de la grille sont atteints sur un trajet $SO \rightarrow NE$ (et pas seulement ceux imposés par la mission).

Il apparaît que `ASFDP`, contrairement à `PSFDP`, récolte toutes les récompenses du problème, c'est-à-dire qu'il ne se limite pas aux buts imposés. Ainsi, `ASFDP` est plus opportuniste que `PSFDP`, mais au détriment du temps de calcul, comme le montre la



Départ → Arrivée (nombre d'états par région carrée, nombre de régions)

FIG. 9.11 – Mode opportuniste : comparaison de l'expansion du sous-espace atteignable $\mathcal{W}$ entre PSFDP et ASFDP, pour des grilles concentriques de différentes tailles et des trajets différents, en imposant le sous-but NE (cf. figure 9.9)

figure 9.13 : ASFDP se comporte quasiment comme SPUDD alors que PSFDP est toujours bien plus rapide. Par conséquent, contrairement à ASFDP, PSFDP est entièrement focalisé sur les sous-buts imposés, en ce sens qu'il n'essaie pas d'atteindre les sous-buts qui ne sont pas imposés par la mission.

Par ailleurs, lorsque tous les buts possibles de la grille sont imposés (mode optimal), PSFDP est optimal à 2% près. Or, résoudre le MDP avec tous les buts possibles imposés, revient à optimiser le MDP entier avec toutes ses récompenses (comme SPUDD et sLAO*). Par conséquent, PSFDP est opportuniste si une partie seulement

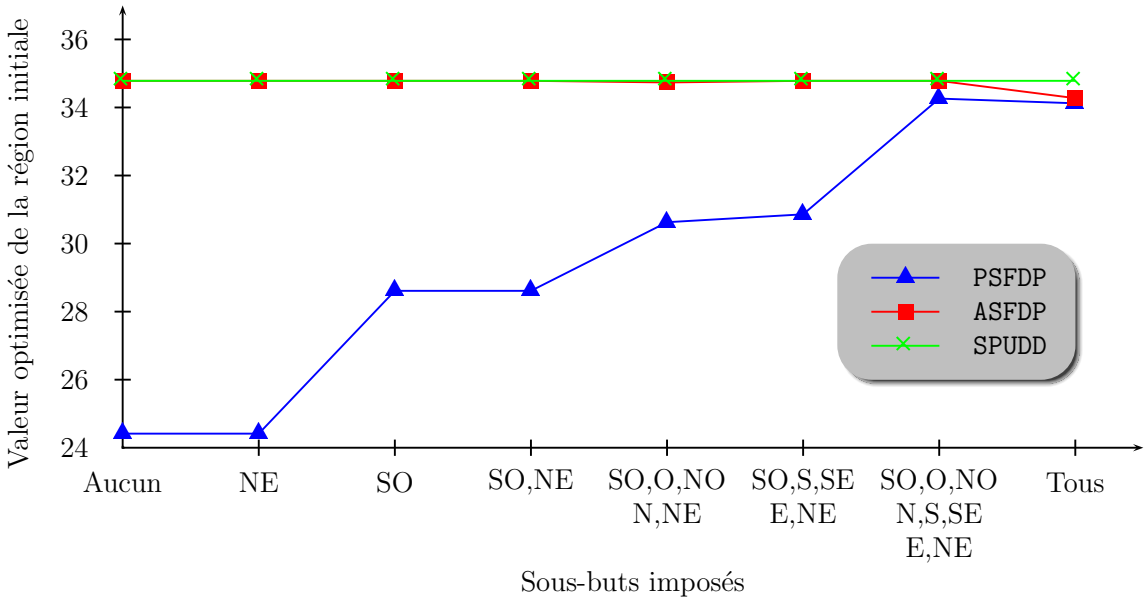


FIG. 9.12 – Du mode opportuniste au mode optimal : comparaison des récompenses accumulées entre PSFDP, ASFDP et SPUDD (optimal) pour une grille concentrique à 9 régions et 9 états par région, et un trajet $SO \rightarrow NE$ (cf. figure 9.9)

des sous-buts est imposée, mais il est optimal et nettement plus performant que SPUDD (cf. figure 9.13) lorsque tous les sous-buts sont imposés.

Mode opportuniste : PSFDP est beaucoup plus rapide, mais moins opportuniste, que ASFDP car, contrairement à ce dernier, il exploite la politique heuristique initiale de plus sûr chemin stochastique, mais il se focalise *uniquement* sur les sous-buts imposés par la mission.
Mode optimal : si toutes les récompenses du MDP sont imposées, PSFDP est optimal et nettement plus rapide que ASFDP et SPUDD.

Condition d'arrêt faible, différentes types d'expansion

Nous avons ensuite testé l'influence des différents types d'expansion pour la condition d'arrêt **faible** ($\mathcal{W} \cap \mathcal{B} \neq \emptyset$). Les algorithmes testés sont donc PsfDP et AsfDP. Comme nous allons le voir, la différence entre les deux types d'expansion est nettement moins marquée que dans les tests précédents, ce qui montre que **la condition d'arrêt influence la sensibilité de sfDP au type d'expansion utilisé**.

Commençons par étudier le mode opportuniste de PsfDP et AsfDP. La figure 9.14 représente le temps de calcul de la stratégie partielle pour des missions de tailles différentes, des trajets tous diagonaux (de la région SO à la région NE), et consistant

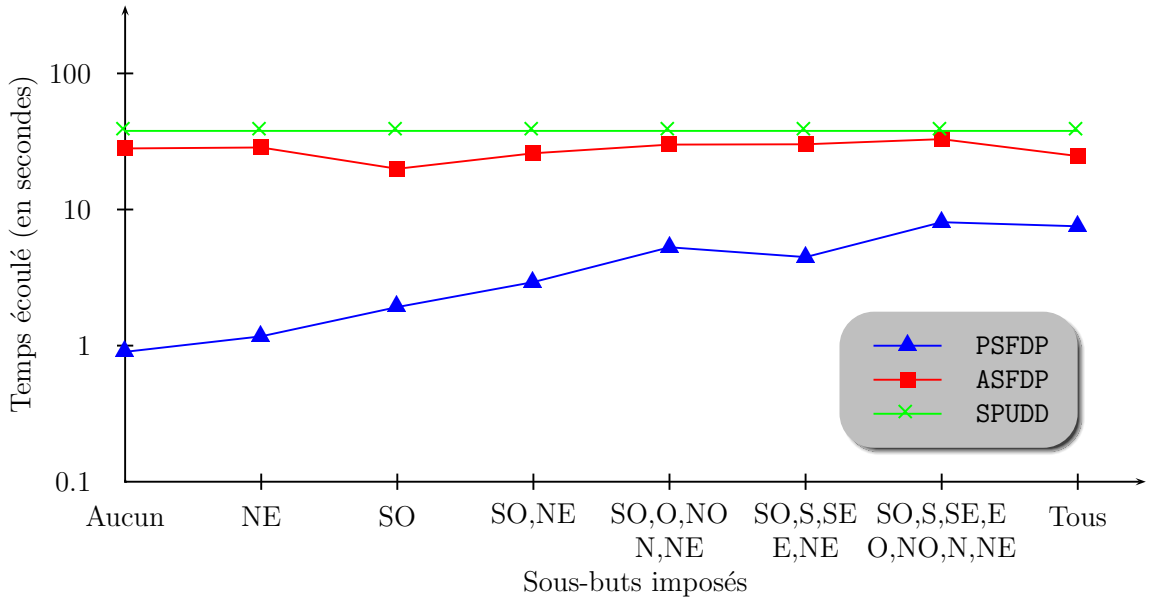


FIG. 9.13 – Du mode opportuniste au mode optimal : comparaison du temps de calcul entre `PSFDP`, `ASFDP` et `SPUDD` (optimal), pour une grille concentrique à 9 régions et 9 états par région, un trajet $SO \rightarrow NE$ et un nombre croissant de sous-buts imposés (cf. figure 9.9)

à atteindre au moins la récompense NE (sous-but imposé). Nous remarquons que la complexité est toujours exponentielle en le nombre de régions, puisqu'il y a autant de variables sous-buts que le nombre de régions et que la complexité de la programmation dynamique symbolique est exponentielle en le nombre de variables d'état [26, 31]. De plus, `PsfDP` reste plus rapide que `AsfDP`, mais la différence est faible, par rapport aux mêmes tests réalisés avec la condition d'arrêt **forte** (cf. figure 9.10).

Afin de comprendre ce phénomène, nous avons étudié le pourcentage d'états atteignables ainsi que son évolution en fonction de la taille de la mission (cf. figure 9.15). Contrairement aux résultats obtenus avec la condition d'arrêt **forte** (cf. figure 9.11), le sous-espace atteignable développé par `AsfDP` est maintenant négligeable devant l'espace entier. De plus, son augmentation durant l'optimisation partielle du MDP est plus importante.

Ainsi, le sous-espace atteignable développé est beaucoup moins important avec la condition d'arrêt **faible**. Ceci se comprend intuitivement grâce à la figure 8.2 (page 199) : avec une condition d'arrêt **faible**, la recherche des sous-buts s'arrête dès qu'un état atteignable au moins satisfait tous les sous-buts imposés. Au contraire, avec condition d'arrêt **forte**, l'expansion du sous-espace atteignable s'arrête lorsque tous les états satisfaisant les sous-buts imposés sont atteignables. Par conséquent,

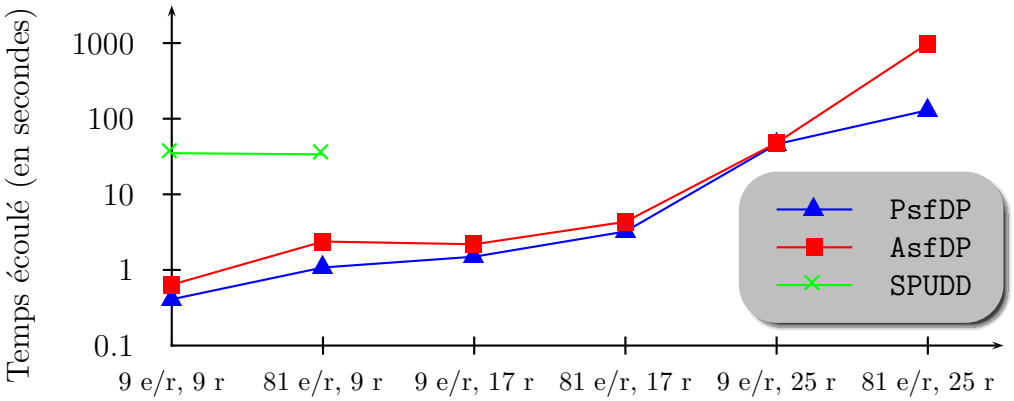


FIG. 9.14 – Mode opportuniste : comparaison du temps de calcul entre `PsfDP`, `AsfDP` et `SPUDD` (en secondes avec un processeur P4 de 2,8 GHz), pour des grilles concentriques de différentes tailles et des trajets $SO \rightarrow NE$, en imposant le sous-but NE (cf. figure 9.9)

le développement du sous-espace atteignable s'arrête plus tôt avec une condition d'arrêt **faible**, si bien qu'il est plus rapide.

En outre, comme le sous-espace atteignable après calcul de l'heuristique est plus petit, son augmentation *relative* avec `AsfDP` est plus importante (cf. figure 9.11). En revanche, cet argument ne tient pas pour `PsfDP` car les états atteignables sont développés en suivant la politique courante, et non de manière isotrope en appliquant toutes les actions possibles. Nous pensons que sa diminution relative moins importante (cf. figure 9.11) est due à l'attraction des états sous-buts non explorés durant le calcul de l'heuristique, car `PsfDP` est attiré par les récompenses en bordure du sous-espace atteignable durant l'optimisation partielle du MDP (cf. figure 9.2 page 223).

D'autre part, la figure 9.16 montre que `PsfDP` est toujours moins opportuniste que `AsfDP`, mais dans des proportions moindres qu'avec une condition d'arrêt **forte** (cf. figure 9.12). En particulier, `AsfDP` ne récolte pas toutes les récompenses du MDP (contrairement à `ASFDP`), même s'il en atteint un peu plus que `PsfDP`. Ce résultat est intéressant car `AsfDP` est à peine plus lent que `PsfDP`, comme le montre la figure 9.17. Cette figure montre également que le temps de calcul augmente moins vite que les récompenses accumulées.

En mode optimal (toutes les récompenses sont imposées), `PsfDP` et `AsfDP` sont optimaux à 2% près. Ce résultat est très satisfaisant car le temps de calcul de `AsfDP` et surtout de `PsfDP` en mode optimal est bien plus faible que celui de `SPUDD` (cf. figure 9.17).

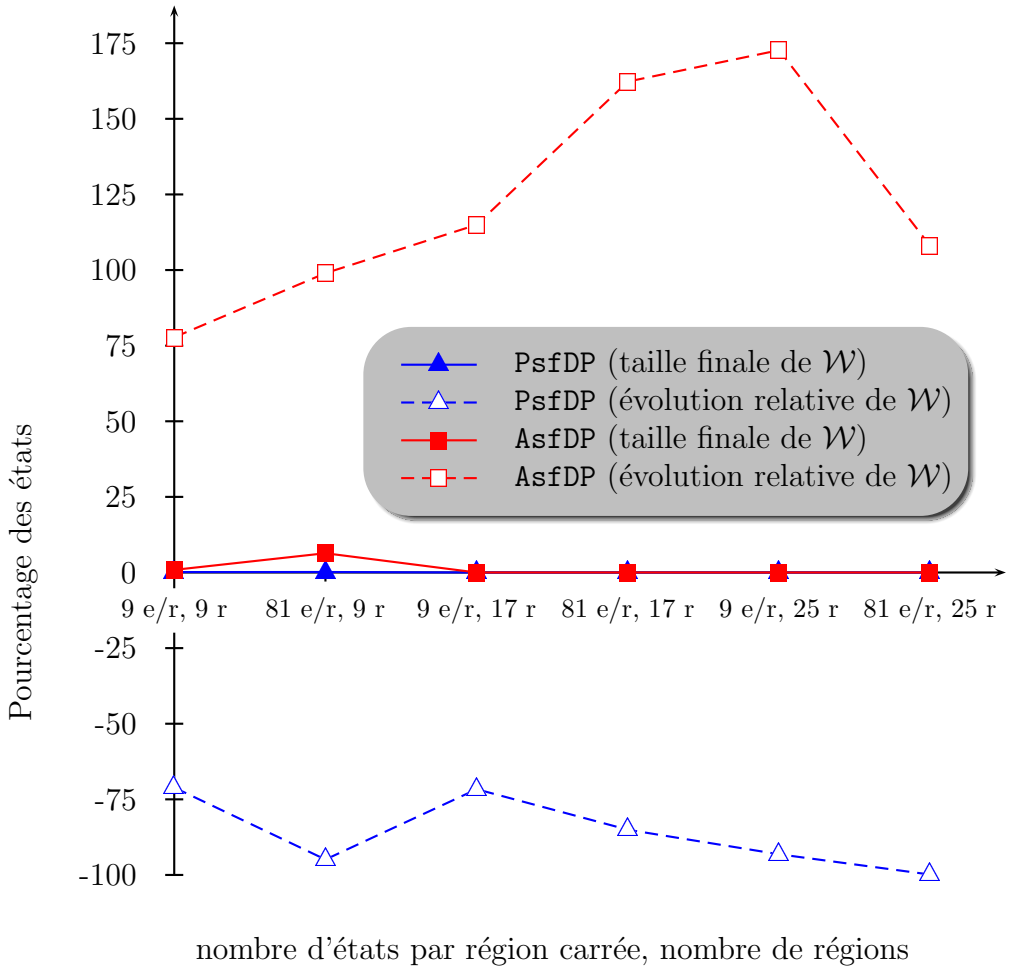


FIG. 9.15 – Mode opportuniste : comparaison de l'expansion du sous-espace atteignable $\mathcal{W}$ entre `PsfDP` et `AsfDP`, pour des grilles concentriques de différentes tailles et des trajets $SO \rightarrow NE$, en imposant le sous-but NE (cf. figure 9.9)

Mode opportuniste : `PsfDP` est un peu plus rapide que `AsfDP` mais moins opportuniste, car l'arrêt précoce de l'expansion du sous-espace atteignable limite les états explorés par `AsfDP`, sans l'empêcher toutefois de découvrir des récompenses autres que les sous-buts imposés.
Mode optimal : `AsfDP` et, surtout, `PsfDP` sont optimaux et nettement plus rapides que `SPUDD`.

Expansion toutes_actions, différentes conditions d'arrêt

Étudions maintenant l'influence de la condition d'arrêt pour un même type d'expansion, en commençant par le type **toutes_actions**. Les algorithmes comparés dans ce paragraphe sont donc `AsfDP` et `ASFDP`.

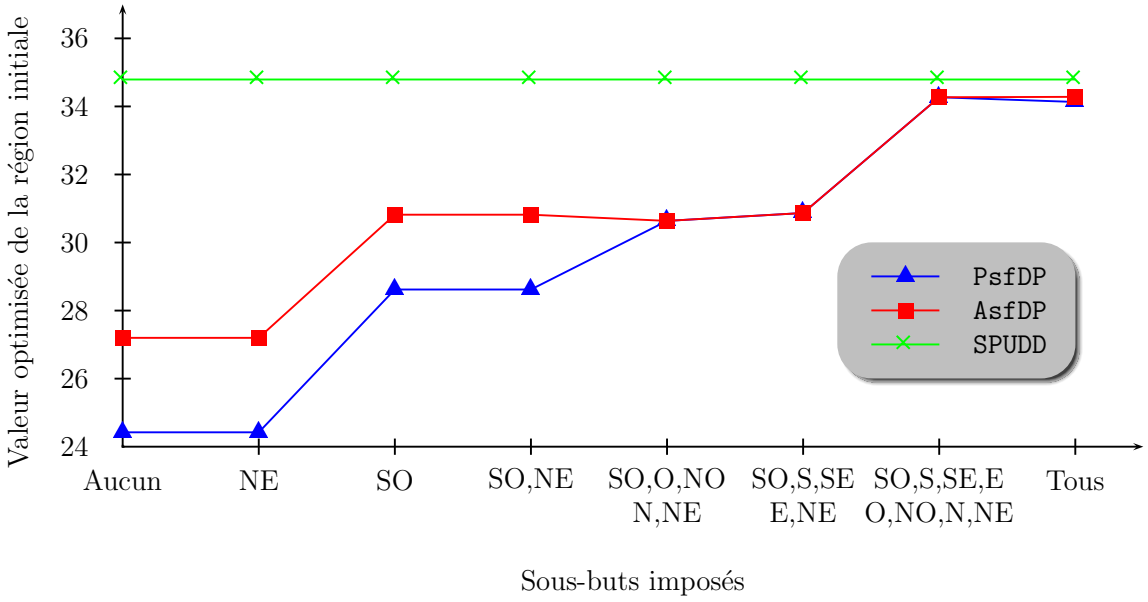


FIG. 9.16 – Du mode opportuniste au mode optimal : comparaison des récompenses accumulées entre `PsfDP`, `AsfDP` et `SPUDD` (optimal), pour une grille concentrique à 9 régions et 9 états par région, un trajet $SO \rightarrow NE$ et un nombre croissant de sous-buts imposés (cf. figure 9.9)

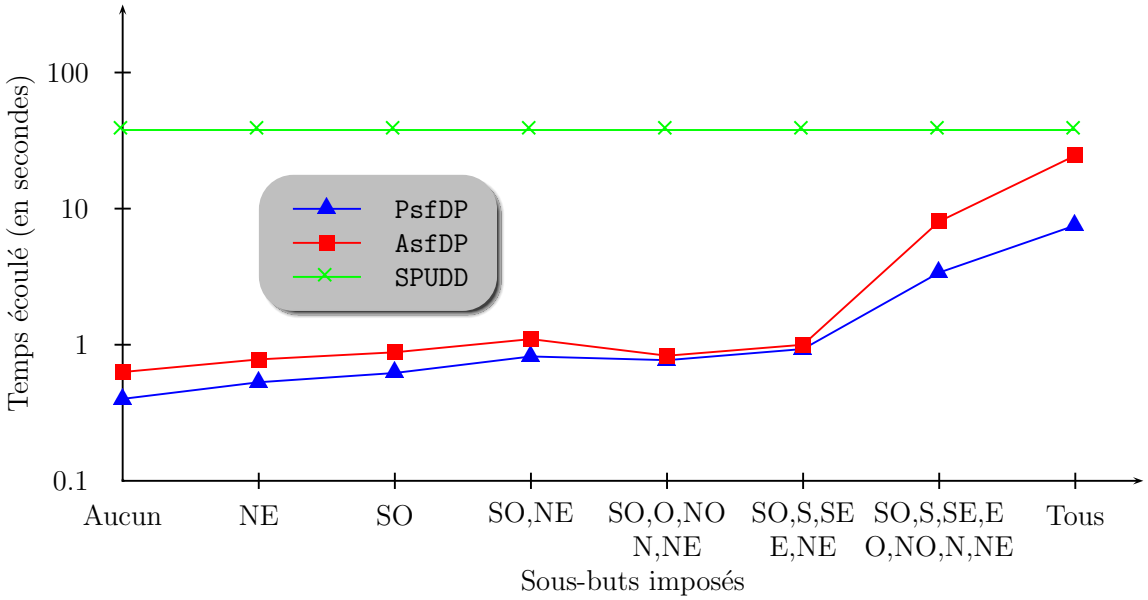


FIG. 9.17 – Du mode opportuniste au mode optimal : comparaison du temps de calcul entre `PsfDP`, `AsfDP` et `SPUDD` (optimal), pour une grille concentrique à 9 régions et 9 états par région, un trajet $SO \rightarrow NE$ et un nombre croissant de sous-buts imposés (cf. figure 9.9)

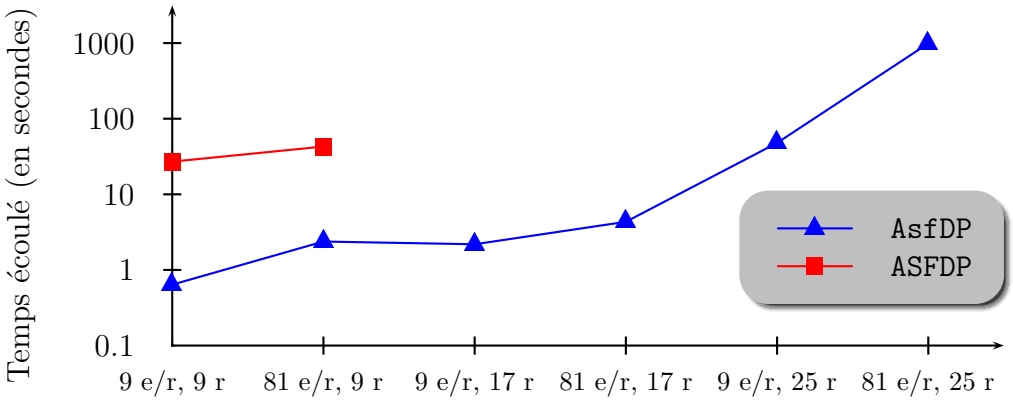


FIG. 9.18 – Mode opportuniste : comparaison du temps de calcul entre `AsfDP` et `ASFDP` (en secondes avec un processeur P4 de 2,8 GHz), pour des grilles concentriques de différentes tailles et des trajets $SO \rightarrow NE$, en imposant le sous-but NE (cf. figure 9.9)

La figure 9.18 montre que `AsfDP` peut résoudre des problèmes beaucoup plus grands que `ASFDP`, et qu'il est entre 10 et 100 fois plus rapide sur les problèmes que `ASFDP` parvient à résoudre.

Ceci s'explique grâce à la figure 9.19, qui montre que `AsfDP` explore quasiment aucun état quand `ASFDP` en explore plus de la moitié. Pourtant, les courbes en pointillés confirment, dans les deux cas, que l'expansion **toutes_actions** a tendance à augmenter sensiblement la taille du sous-espace atteignable. Par conséquent, la condition d'arrêt prédomine sur le type d'expansion en termes de complexité et de coût de calcul.

D'autre part, la figure 9.20 montre que `ASFDP` est nettement plus opportuniste (et même trop) que `AsfDP`, car `ASFDP` récolte par défaut toutes les récompenses du problème. La condition d'arrêt **faible** limite donc la découverte opportuniste des récompenses du MDP, car elle arrête très tôt l'expansion du sous-espace atteignable. D'ailleurs, la figure 9.21 montre que `ASFDP` est comparable à `SPUDD`, alors qu'ils sont beaucoup plus lents que `AsfDP`.

En mode optimal, `ASFDP` et `AsfDP` sont tous les deux optimaux à 2% près. Néanmoins, leurs temps de calcul sont du même ordre de grandeur que celui de `SPUDD`, si bien qu'ils n'ont aucun intérêt dans ce mode.

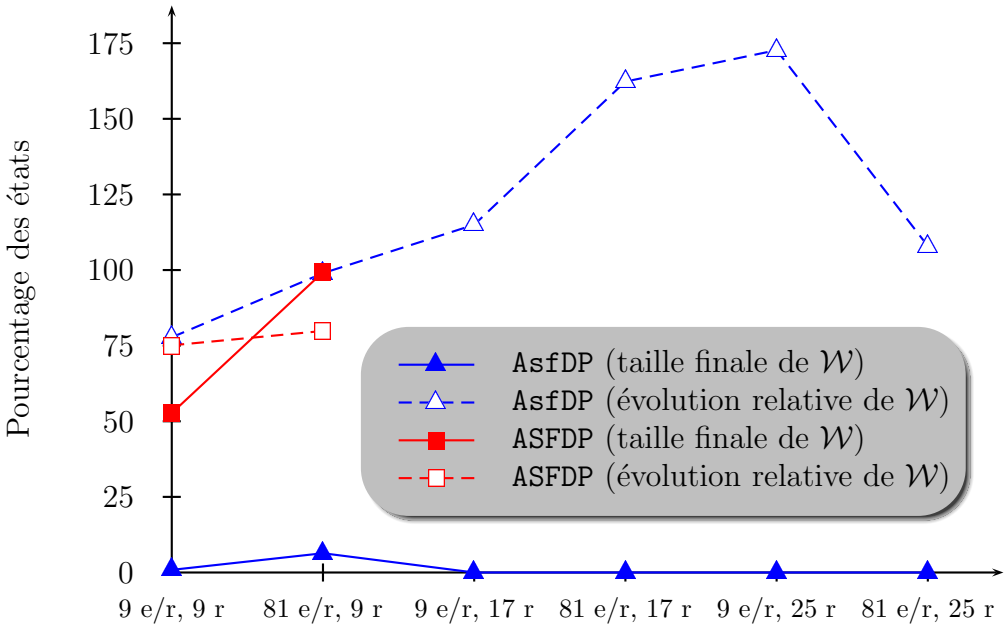


FIG. 9.19 – Mode opportuniste : comparaison de l'expansion du sous-espace atteignable $\mathcal{W}$ entre `AsfDP` et `ASFDP`, pour des grilles concentriques de différentes tailles et des trajets $SO \rightarrow NE$, en imposant le sous-but NE (cf. figure 9.9)

Mode opportuniste : `AsfDP` est plus adapté que `ASFDP`, car `AsfDP` est beaucoup plus rapide et économe que `ASFDP`, qui ne se focalise absolument pas sur les sous-buts imposés.
Mode optimal : `AsfDP` et `ASFDP` sont tous les deux optimaux, mais leurs temps de calcul sont comparables à `SPUDD`.

Expansion suivre_politique, différentes conditions d'arrêt

Étudions enfin l'influence du type d'expansion **suivre_politique**, ce qui revient à comparer les algorithmes `PsfDP` et `PSFDP`.

La figure 9.22 montre que `PsfDP` est plus rapide que `PSFDP`, et que la différence s'accroît avec la taille des problèmes. De plus, contrairement à `PSFDP`, `PsfDP` peut résoudre le problème le plus difficile testé. Ainsi, en mode opportuniste, `PsfDP` est plus intéressant que `PSFDP`, si le temps de calcul est une ressource limitée.

Il apparaît sur la figure 9.23 que les sous-espaces atteignables en sortie des algorithmes `PsfDP` et `PSFDP` ont la même taille, alors que la régression de la taille de ces sous-espaces est nettement plus importante avec `PSFDP`. Ainsi, ce dernier perd

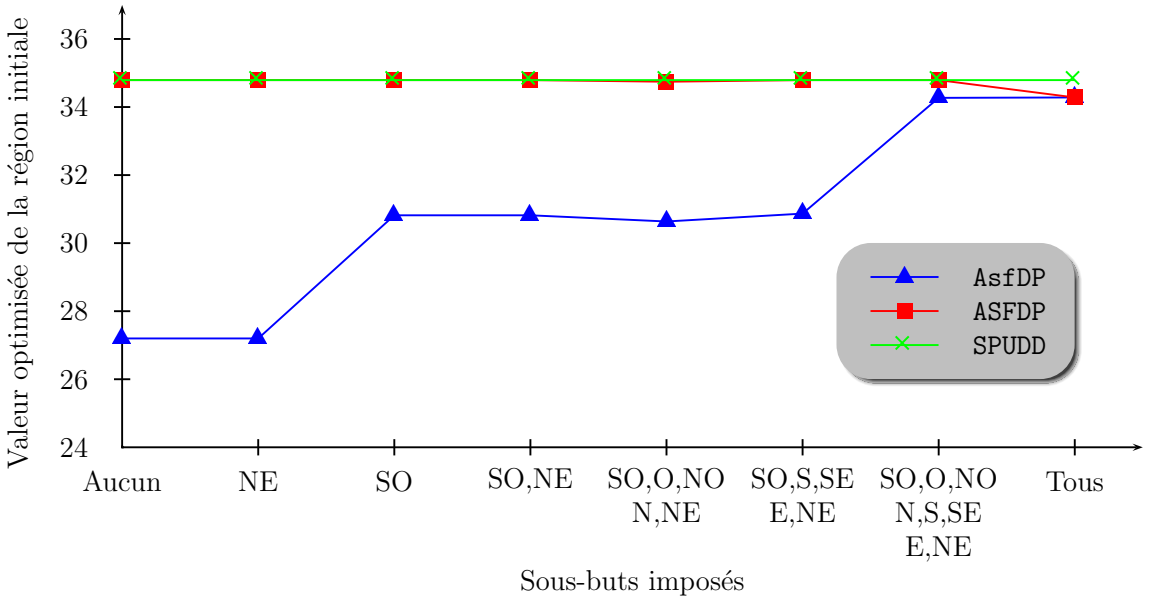


FIG. 9.20 – Du mode opportuniste au mode optimal : comparaison des récompenses accumulées entre `AsfDP`, `ASFDP` et `SPUDD` (optimal) pour une grille concentrique à 9 régions et 9 états par région, et un trajet $SO \rightarrow NE$ (cf. figure 9.9)

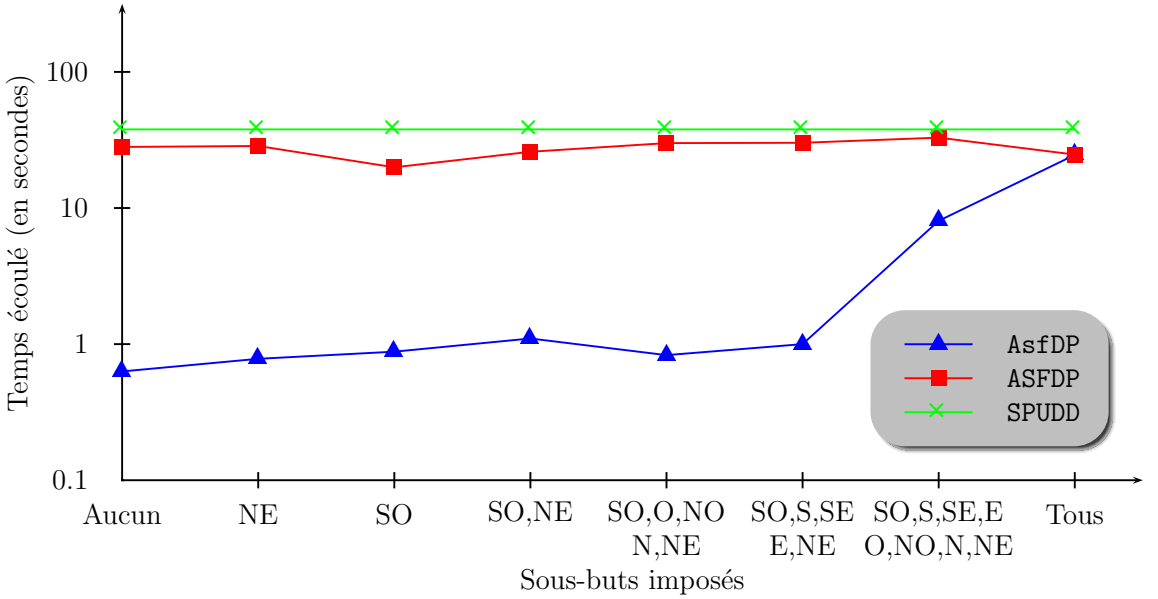


FIG. 9.21 – Du mode opportuniste au mode optimal : comparaison du temps de calcul entre `AsfDP`, `ASFDP` et `SPUDD` (optimal), pour une grille concentrique à 9 régions et 9 états par région, un trajet $SO \rightarrow NE$ et un nombre croissant de sous-buts imposés (cf. figure 9.9)

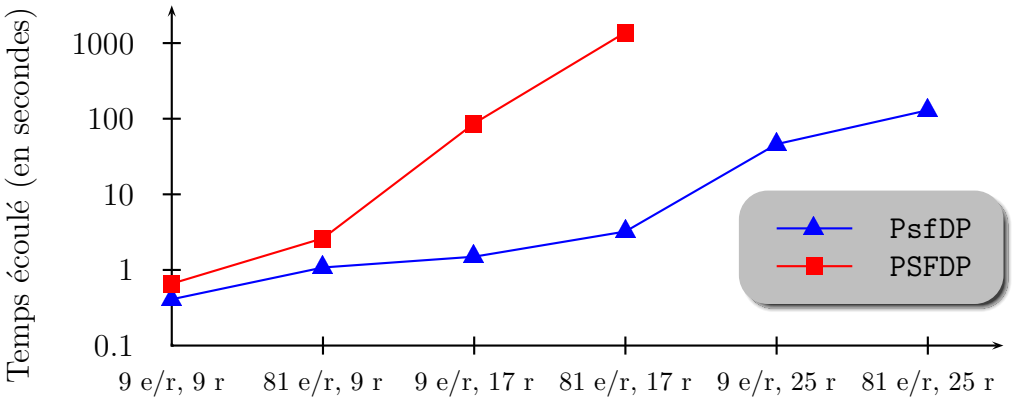


FIG. 9.22 – Mode opportuniste : comparaison du temps de calcul entre `PsfDP` et `PSFDP` (en secondes avec un processeur P4 de 2,8 GHz), pour des grilles concentriques de différentes tailles et des trajets $SO \rightarrow NE$, en imposant le sous-but NE (cf. figure 9.9)

du temps à réduire le sous-espace atteignable, qui est initialement plus grand que celui de `PsfDP` à cause de la condition d'arrêt **forte** (cf. figure 8.2 page 199), ce qui explique que `PsfDP` est plus rapide que `PSFDP`.

De plus, ce résultat prouve aussi que les cas supplémentaires de satisfaction des sous-buts, pris en compte par la condition d'arrêt **forte**, ne participent pas à l'amélioration de la stratégie partielle. Ceci est entièrement confirmé par le figure 9.24, qui prouve que `PsfDP` et `PSFDP` récoltent exactement le même nombre de récompenses. Toutefois, `PsfDP` est meilleur que `PSFDP` car son temps de calcul est inférieur (cf. figure 9.25). Ainsi, pour le type d'expansion **suivre_politique**, les états buts explorés par la condition d'arrêt **faible** suffisent à l'optimisation de la stratégie.

Enfin, en mode optimal, la figure 9.24 montre que `PsfDP` et `PSFDP` sont optimaux à 2% près. De plus, ils sont beaucoup plus rapides que `SPUDD`, mais tous les deux identiques. Ainsi, `PsfDP` et `PSFDP` sont équivalents en mode optimal.

Mode opportuniste : `PsfDP` est meilleur que `PSFDP` car, à fonctions de valeur égales, `PsfDP` est plus rapide et plus focalisé que `PSFDP`.
Mode optimal : `PsfDP` et `PSFDP` sont équivalents, et nettement plus rapides que `SPUDD`.

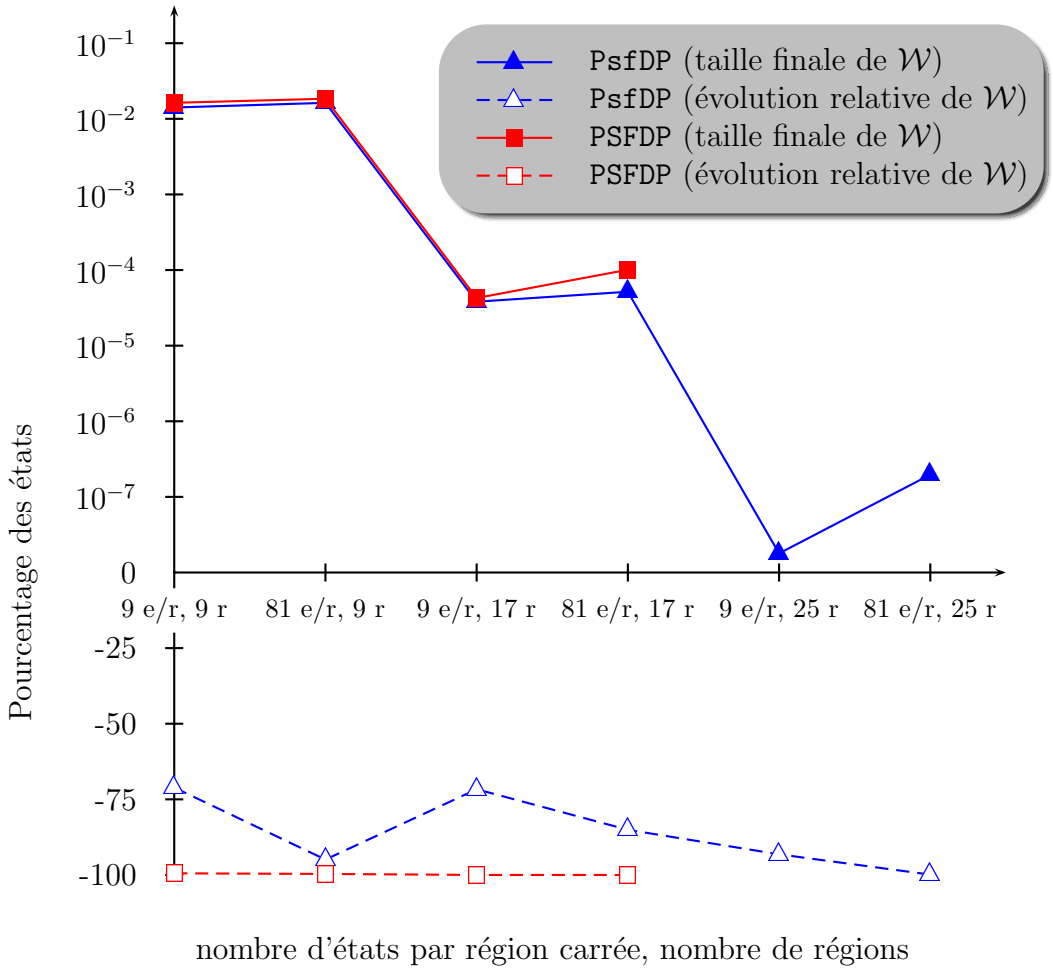


FIG. 9.23 – Mode opportuniste : comparaison de l'expansion du sous-espace atteignable $\mathcal{W}$ entre `PsfDP` et `PSFDP`, pour des grilles concentriques de différentes tailles et des trajets $SO \rightarrow NE$, en imposant le sous-but NE (cf. figure 9.9)

Bilan comparatif

Récapitulons les résultats que nous avons obtenu en mode opportuniste :

- `PSFDP` est beaucoup plus rapide que `ASFDP`, mais nettement moins opportuniste ;
- `PsfDP` est légèrement plus rapide que `AsfDP`, mais un peu moins opportuniste ;
- `AsfDP` est beaucoup plus rapide que `ASFDP`, mais nettement moins opportuniste ;
- `PsfDP` est nettement plus rapide que `PSFDP`, et autant opportuniste.

`ASFDP` est l'algorithme le plus opportuniste, car il récolte toutes les récompenses du MDP, indépendemment de celles qui lui sont imposées. Cependant, ceci ne corres-

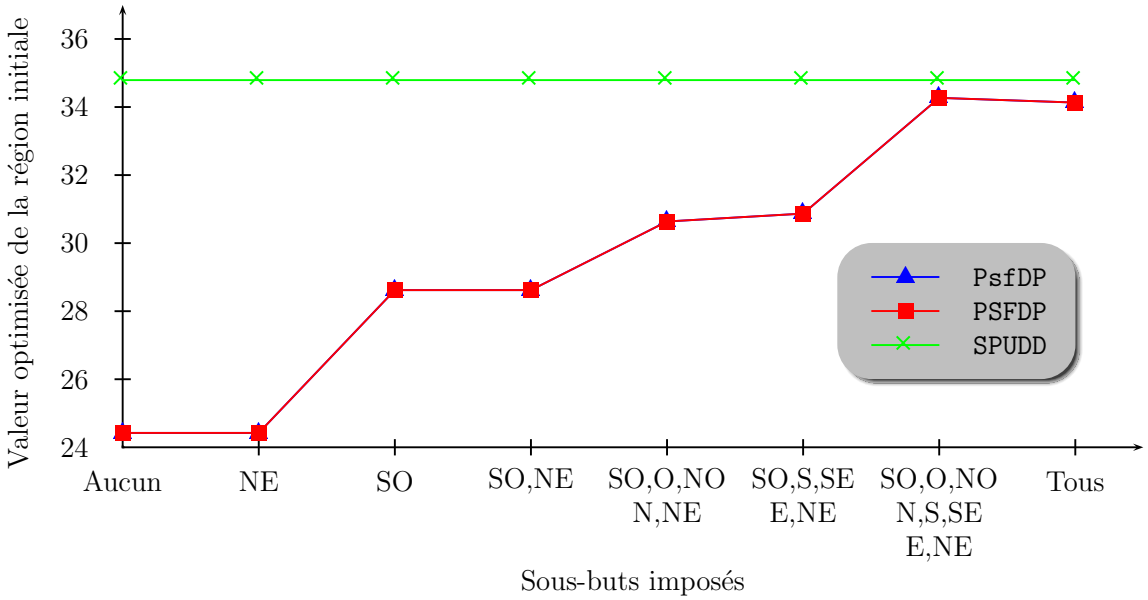


FIG. 9.24 – Du mode opportuniste au mode optimal : comparaison des récompenses accumulées entre `PsfDP`, `PSFDP` et `SPUDD` (optimal) pour une grille concentrique à 9 régions et 9 états par région, et un trajet $SO \rightarrow NE$ (cf. figure 9.9)

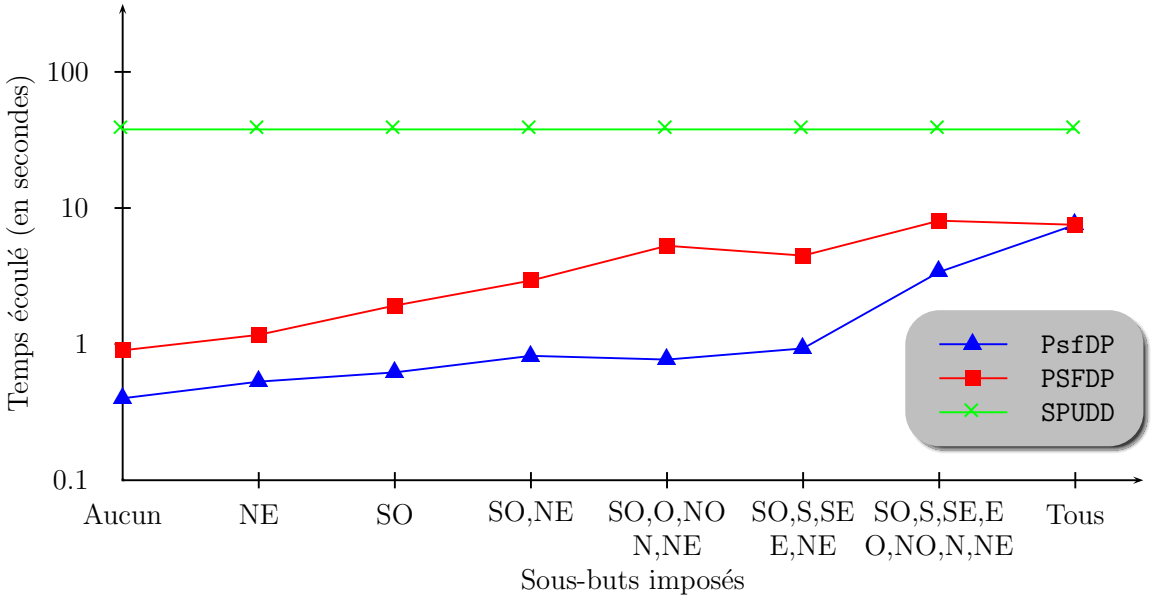


FIG. 9.25 – Du mode opportuniste au mode optimal : comparaison du temps de calcul entre `PsfDP`, `PSFDP` et `SPUDD` (optimal), pour une grille concentrique à 9 régions et 9 états par région, un trajet $SO \rightarrow NE$ et un nombre croissant de sous-buts imposés (cf. figure 9.9)

pond pas nécessairement à ce que nous attendons d'un algorithme opportuniste : si nous souhaitons obtenir toutes les récompenses indépendemment de certains sous-buts imposés, autant résoudre directement le MDP en mode optimal. D'ailleurs, nous avons vu que `ASFDP` est comparable à `SPUDD`, y compris en mode opportuniste. Pour cette raison, `ASFDP` ne nous semble pas être un bon algorithme opportuniste.

En outre, en mode opportuniste, `PSFDP` n'a aucun avantage par rapport à `PsfDP`, car il est beaucoup plus lent tout en récoltant les mêmes récompenses. Nous pouvons donc l'éliminer si bien qu'il nous reste les algorithmes `PsfDP` et `AsfDP`. Si le temps de calcul est plus important que les récompenses accumulées (l'opportunisme), `PsfDP` devrait être préféré à `AsfDP`, et vice-versa.

En mode optimal, tous les algorithmes sont optimaux à 2% près (sur les problèmes testés) si bien que le choix du meilleur algorithme porte sur le plus rapide. Pour un même type d'expansion, les algorithmes sont équivalents : `AsfDP` et `ASFDP` d'un part, et `PsfDP` et `PSFDP` d'autre part, sont identiques en mode optimal. Néanmoins, l'expansion **toutes_actions** est toujours plus lente que l'expansion **suivre_politique**. Ainsi, `PsfDP` et `PSFDP` sont les meilleurs algorithmes optimaux testés. Aussi, comme `PsfDP` est meilleur que `PSFDP` en mode opportuniste, nous retiendrons l'algorithme `PsfDP`.

Par conséquent, nous arrivons au constat suivant, qui est finalement intuitif au regard des différents types d'expansion (cf. figure 9.2 page 223) et de condition d'arrêt (cf. figure 8.2 page 199).

Mode opportuniste : Si l'opportunisme est un critère prédominant, `AsfDP` est le meilleur algorithme opportuniste, avec toutefois une préférence pour `PsfDP` si la rapidité de calcul est plus importante que l'opportunisme.
Mode optimal : `PsfDP` et `PSFDP` sont les meilleurs algorithmes testés, avec une préférence pour `PsfDP` qui est plus performant en mode opportuniste.

Replanification en ligne

Les tests précédents ont montré la supériorité de `AsfDP` et `PsfDP` en mode opportuniste, ce dernier étant meilleur en mode optimal. Cependant, même si ces deux algorithmes sont nettement plus performants que `SPUDD`, leur complexité reste exponentielle en le nombre de sous-buts imposés (droites en échelle logarithmique).

Aussi, il n'est pas raisonnable d'utiliser `AsfDP` ou `PsfDP` en ligne : à chaque fois qu'un nouveau sous-but est imposé, ces deux algorithmes replanifient entièrement la stratégie depuis les états initiaux, indépendemment du sous-espace déjà atteint et de la politique courante. Pourtant, tous les sous-buts imposés (le nouveau et les précédents) doivent être à chaque fois pris en compte, car l'agent n'a pas nécessairement atteint tous les sous-buts précédemment imposés lorsque la nouvelle requête de sous-but survient. Par conséquent, les complexités de `AsfDP` et `PsfDP`

augmentent exponentiellement au fur et à mesure que les sous-buts sont imposés durant la mission.

C'est sur ce constat que nous avons développé la classe d'algorithmes incrémentaux `IsfDP`. Afin d'évaluer les performances de nos algorithmes incrémentaux, nous avons simulé une mission de déplacement et de prise d'information dans une grille concentrique à 9 régions et 9 états par région (cf. figure 9.9), au cours de laquelle les récompenses imposées au drone sont ajoutées de manière incrémentale. Comme nous avons vu dans le paragraphe précédent que la condition d'arrêt **faible** est meilleure que la condition d'arrêt **forte**, nous avons testé les algorithmes suivants :

- `AsfDP` et `PsfDP` : replanifient toute la stratégie à chaque fois qu'un nouvelle récompense est imposée ;
- `IAsfDP` et `IPsfDP` : replanifient la stratégie partiellement, en utilisant la stratégie optimisée et le sous-espace atteignable obtenus sur la base des précédentes récompenses imposées (cf. figure 9.4).

La figure 9.26 représente le temps de calcul nécessaire pour replanifier la nouvelle stratégie dès qu'une nouvelle récompense est imposée. L'échelle en temps de calcul étant logarithmique, il apparaît clairement que le temps de replanification de la stratégie avec `AsfDP` ou `PsfDP` est exponentiel en le nombre de sous-buts imposés, ce dernier étant néanmoins beaucoup plus rapide. Au contraire, le temps de replanification avec `IPsfDP` est quasiment constant à chaque replanification, sauf lors de l'ajout de la récompense SE. La figure 9.27 montre également que `IPsfDP` permet de récolter autant de récompenses que `PsfDP`, ce qui prouve la supériorité de l'approche incrémentale pour la replanification en ligne (du moins pour le type d'expansion **suivre_politique**). Aussi, lorsque toutes les récompenses ont été imposées, `IPsfDP` est optimal à 3% près.

En ce qui concerne le type d'expansion **toutes_actions**, la figure 9.27 montre que `IAsfDP` récolte beaucoup plus de récompenses que `AsfDP`. Il se comporte d'ailleurs comme `SPUDD`, si bien qu'il n'a aucun intérêt pour la replanification en ligne. Son temps de calcul est le pire de tous à part `SPUDD`, ce qui prouve qu'il explore beaucoup plus d'états atteignables que `AsfDP`, `PsfDP` ou `IPsfDP`. En effet, `IAsfDP` appelle itérativement `AsfDP`, dont le sous-espace atteignable augmente en suivant toutes les actions possibles indépendemment de la stratégie courante. L'augmentation cumulée du sous-espace atteignable sur chaque requête de sous-buts entraîne rapidement `IAsfDP` à visiter beaucoup plus d'états atteignables que les autres algorithmes (`SPUDD` mis à part). Au contraire, `IPsfDP` appelle itérativement `PsfDP`, dont le sous-espace atteignable suit la stratégie courante : ainsi, `IPsfDP` se contente d'englober uniquement les sous-buts imposés jusqu'à l'instant présent (cf. figures 9.2 page 223 et 9.4 page 229).

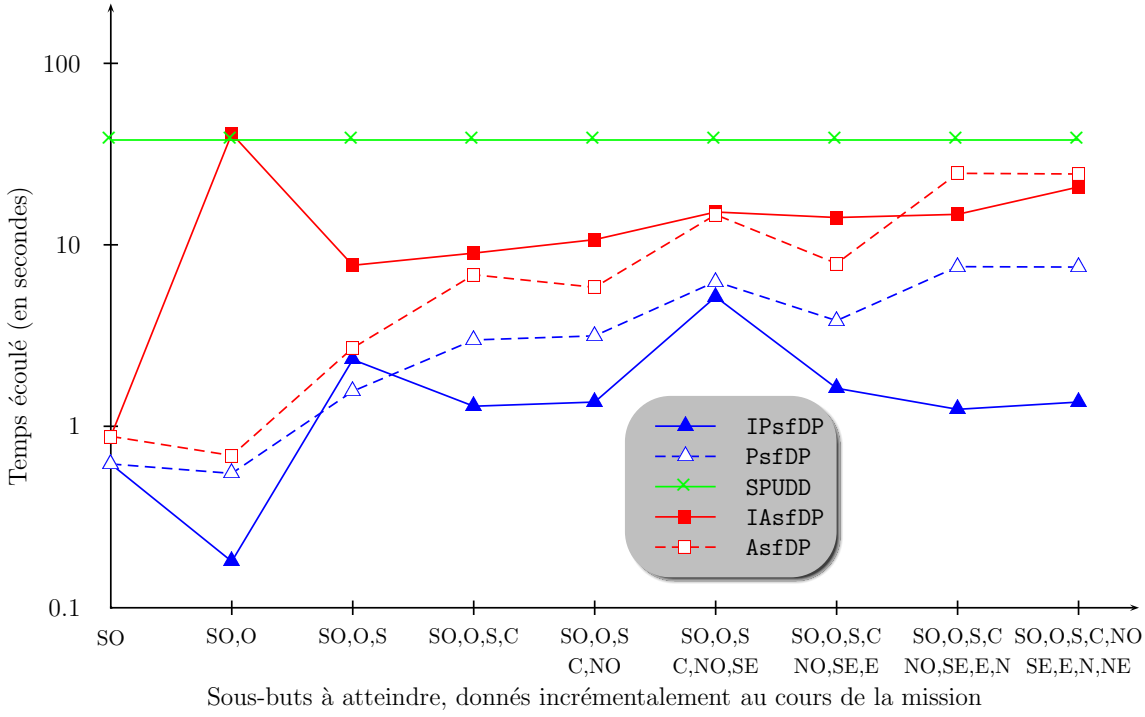


FIG. 9.26 – Comparaison du temps de calcul après chaque replanification entre `IPsfDP`, `PsfDP`, `IAsfDP`, `AsfDP` et `SPUDD` (en secondes avec un processeur P4 de 2,8 GHz), pour une grille concentrique à 9 régions et 9 états par région carrée, un trajet $SO \rightarrow NE$, et un nombre croissant de sous-buts imposés durant la mission (cf. figure 6.24 page 172)

`IPsfDP` est le meilleur algorithme de replanification en ligne testé : il est optimal lorsque toutes les récompenses ont été imposées, et le temps de recalcul de la stratégie est quasiment constant à chaque nouvelle requête de sous-but.

9.3.3 Intégration à l'outil de modélisation, résolution et simulation de missions ReSSAC

Nous avons intégré la classe d'algorithmes `sfDP` à l'outil de modélisation, résolution et simulation ReSSAC (cf. chapitre 6 page 153).

Les sous-ensembles d'états initiaux ou buts sont des formules logiques de la variable de région R et des variables d'objectifs $(O^r)_{r \in V_R}$ (chaque région contient un

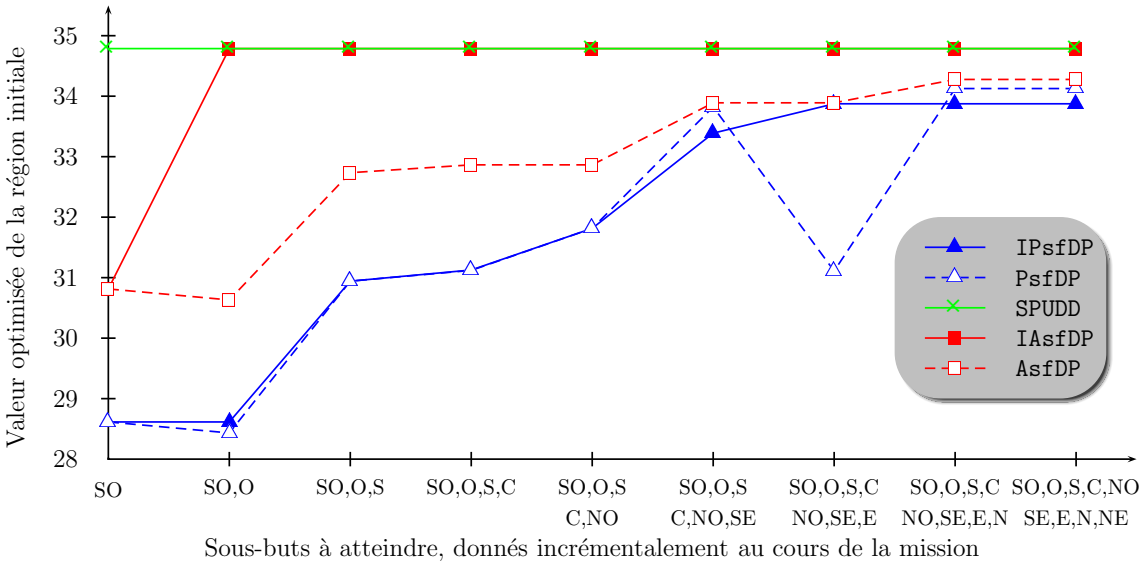


FIG. 9.27 – Comparaison des récompenses accumulées après chaque replanification entre `IPsfDP`, `PsfDP`, `IAsfDP`, `AsfDP` et `SPUDD` pour une grille concentrique à 9 régions et 9 états par région, un trajet $SO \rightarrow NE$, et un nombre croissant de sous-buts imposés par la mission (cf. figure 6.24 page 172)

objectif possible à atteindre) de la forme suivante :

$$\mathcal{B}, \mathcal{I} = \left(\bigwedge_{k=1}^{q} (O^{r_{i_k}} = \textbf{atteint}) \right) \wedge \left(\bigvee_{k=1}^{r} (R = r_{i_k}) \right)$$

Ainsi, le drone part d'une région inconnue au moment de l'optimisation de la stratégie, mais qui appartient à $(r_{i_k})_{1 \leqslant k \leqslant r}$, avec un ensemble d'objectifs $(O^{r_{i_k}})_{1 \leqslant k \leqslant q}$ qu'il n'est pas obligatoire d'atteindre (ce qui revient à mettre les valeurs correspondantes à la valeur **atteint** pour le planificateur).

En ce qui concerne le sous-ensemble de buts, le drone doit arriver dans une des régions $(r_{i_k})_{1 \leqslant k \leqslant r}$ en ayant atteint les objectifs $(O^{r_{i_k}})_{1 \leqslant k \leqslant q}$. Ainsi, les sous-ensemble initiaux et buts sont des conjonctions de choix, où seul le choix de la région de départ ou d'arrivée est varié.

La figure 9.28 représente le choix des régions possibles et des objectifs imposés pour les sous-ensembles d'états initiaux et buts dans l'outil de missions RESSAC. Bien entendu, ces contraintes de planification ne sont prises en compte que si l'algorithme d'optimisation du MDP factorisé hiérarchique est différent de `SPUDD`. La figure 9.29 représente l'évolution du sous-espace atteignable durant l'optimisation partielle de la stratégie.

La classe d'algorithmes incrémentaux de `sfDP` s'applique, puisque le sous-ensemble

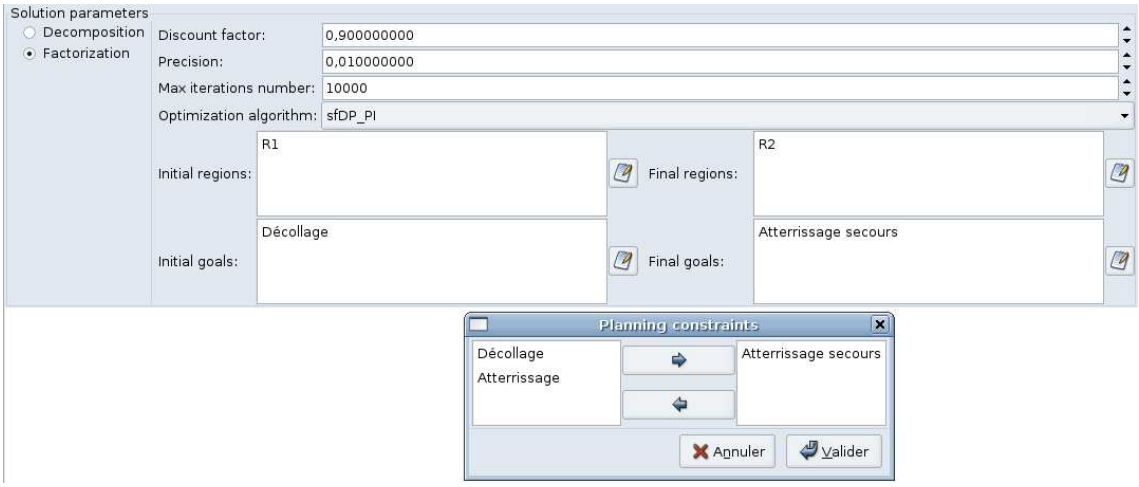


FIG. 9.28 – Capture d'écran de l'éditeur de régions possibles et d'objectifs imposés au planificateur pour les sous-ensembles d'états initiaux et buts

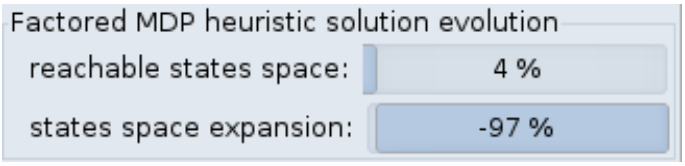


FIG. 9.29 – Capture d'écran de l'évolution du sous-espace atteignable durant l'optimisation partielle de la stratégie

d'états buts est donné sous la forme d'une conjonction d'objectifs à atteindre et d'un choix de régions d'arrivée (cf. formalisme de `IsfDP` page 229) :

- $\mathcal{B}^1 = \bigcup_{1 \leqslant k \leqslant r} (R = r_{i_k})$
- $\forall\, j > 1\ ,\ \mathcal{B}^j = \mathcal{B}^{j-1} \cap (O^{r_{i_j}} = \textbf{atteint})$

Enfin, la simulation prend en compte les états initiaux, en ce sens que la région de départ est choisie au hasard dans la liste des régions initiales possibles, et les variables d'objectifs initiaux sont fixées à la valeur **atteint**.

Ainsi, notre outil de modélisation, résolution et simulation de missions ReSSAC intègre toutes les techniques mises en œuvre dans cette thèse, mise à part la réduction automatique d'arité des variables d'état (étudiée très récemment).

Bilan

Nous avons proposé une classe d'algorithmes de Programmation Dynamique Symbolique Heuristique et Focalisée, appelée `sfDP`, qui cible la recherche de la stratégie optimale sur les récompenses du MDP. Inspiré schématiquement de `sLAO`*, notre algorithme commence par calculer une heuristique, qui guide ensuite une recherche itérative de la solution : une phase d'expansion du sous-espace d'états atteignables, guidée par l'heuristique, alterne avec une phase d'optimisation du MDP sur les états atteignables courants.

Contrairement à `sLAO`*, l'heuristique de `sfDP` n'est pas une approximation de la fonction de valeur du MDP sur l'ensemble d'états total. Notre heuristique consiste en le calcul d'une politique de plus sûr chemin stochastique, qui maximise la probabilité d'atteindre les récompenses du problème sur un sous-ensemble initial d'états atteignables. Cette heuristique est obtenue en trois temps :

1. calcul des états atteignables depuis les états initiaux jusqu'aux états buts, en rendant toutes les transitions déterministes ;
2. optimisation d'une politique de plus sûr chemin stochastique sur ce sous-ensemble d'états atteignables ;
3. filtrage des états qui ont une faible probabilité de mener aux buts, comparés aux états initiaux.

Notre heuristique fournit dont une politique initiale *et* un sous-ensemble initial d'états atteignables, alors que celle de `sLAO`* consiste seulement en une approximation de la fonction de valeur sur l'ensemble des états du MDP. Cependant, contrairement à `sLAO`*, l'heuristique de `sfDP` n'est pas compatible avec la fonction de valeur du MDP, si bien qu'elle ne peut pas être exploitée durant l'optimisation du MDP. Autrement dit, la valeur des états situés sur la frontière du sous-espace atteignable n'est pas connue, au point que la fonction de valeur n'est pas guidée vers les récompenses extérieures au sous-espace atteignable. Ainsi, nous distinguons deux modes d'utilisation de `sfDP`, dont le deuxième est le cas limite du premier :

– mode *opportuniste* : une partie seulement des récompenses est imposée au planificateur, qui se trouvent nécessairement dans le sous-espace atteignables ; les autres récompenses sont découvertes de manière opportuniste au cours de l'expansion du sous-espace atteignable courant.

– mode *optimal* : toutes les récompenses sont imposées au planificateur ; la stratégie est donc optimisée sur un sous-espace d'états qui contient toutes les récompenses du MDP.

L'avantage de `sfDP` sur `sLAO*` réside dans une exploration de l'espace d'états qui n'est jamais totale. Le calcul de l'heuristique de `sLAO*` nécessite en effet d'évaluer, certes approximativement, tous les états du MDP, alors que le calcul de notre heuristique consiste *d'abord* en l'obtention d'un sous-espace atteignable initial, dans lequel l'heuristique est *ensuite* optimisée.

Néanmoins, lors de la Première Compétition Internationale de Planification Stochastique, notre planificateur s'est avéré moins performant que `sLAO*`, bien que placé troisième dans la catégorie MDP. Toutefois, la version de `sfDP` utilisée durant la compétition était prématurée et boguée. Nous souhaiterions donc comparer de nouveau `sLAO*` à `sfDP` à l'occasion de la prochaine compétition de planification, puisque notre planificateur est désormais mature. Nous n'avons pas pu tester `sLAO*` nous-mêmes, car nous ne disposons pas d'un algorithme tel `APRICODD`, permettant de calculer approximativement la fonction de valeur sur l'ensemble d'états total. Cette implémentation est bien entendu nécessaire afin de comparer extensivement `sLAO*` et `sfDP`.

En outre, nous avons implémenté et testé différentes versions de `sfDP`, qui diffèrent suivant le type d'expansion du sous-espace atteignable et la condition d'arrêt de cette expansion. Les états atteignables peuvent être développés soit en appliquant toutes les actions simultanément, soit en suivant la politique courante, ce qui est plus ciblé vers les récompenses imposées, mais qui est moins opportuniste. La condition d'arrêt est soit faible (au moins un état atteignable est un état but), soit forte (tous les états buts sont atteignables), ce qui permet d'explorer tous les chemins menant aux buts, mais qui développe beaucoup plus d'états atteignables.

Nous avons testé et comparé les différentes versions de `sfDP` sur des missions de déplacement et de prise d'information de type ReSSAC. La condition d'arrêt faible offre de meilleures performances que la condition d'arrêt forte, car elle permet une optimisation plus rapide tout en garantissant la même moyenne de récompenses accumulées durant le processus. Quant au type d'expansion, suivre la politique est meilleur en mode optimal, mais il accumule moins de récompenses en mode opportuniste.

En outre, nous avons développé une version incrémentale de `sfDP`, appelée `IsfDP`, adaptée à la replanification en ligne. L'algorithme `IsfDP` appelle `sfDP` itérativement sur une liste croissante de sous-buts imposés au cours de la mission. Dès qu'une nouvelle requête de sous-buts est envoyée à l'agent, `IsfDP` replanifie la stratégie en utilisant celle optimisée avec les précédents sous-buts imposés, et en développant les nouveau états atteignables depuis le précédent sous-espace atteignable. Des tests sur des missions de type ReSSAC ont montré que `IsfDP`, contrairement à `sfDP`, permet d'améliorer la stratégie en temps quasiment constant.

Conclusion

Dans cette thèse, nous avons étudié des modèles et des algorithmes de planification d'actions en environnement incertain. Plus spécifiquement, nous nous sommes intéressés aux Processus Décisionnels de Markov, qui sont un modèle de planification stochastique proactive : l'optimisation d'un MDP vise à produire une stratégie hors-ligne, optimale au regard d'un critère donné, en supposant que la dynamique du système décisionnel est markovienne, c'est-à-dire sans mémoire du passé.

Nous avons proposé les contributions suivantes :

- **généralisation des techniques de décomposition de MDP énumérés aux MDP factorisés**, menant à un modèle et un algorithme de réduction automatique des variables d'état d'un MDP factorisé (algorithme `SVAR`) ;
- **modèle hiérarchique et générique de Réseau Bayésien Dynamique**, permettant de modéliser facilement des problèmes de grande taille ;
- **algorithme d'instanciation automatique des DBN génériques et hiérarchiques**, connaissant un pool de macro-actions qui peut être obtenu par décomposition de sous-MDP énumérés ;
- **algorithme symbolique d'optimisation heuristique et focalisée des MDP factorisés**, qui cible les récompenses du modèle si des états initiaux possibles sont connus a priori (algorithme `sfDP`) ;
- **version en ligne de l'algorithme** `sfDP`, qui améliore une stratégie optimisée sur une liste incrémentale de récompenses imposées au fur et à mesure du déroulement de la mission.

Nous avons d'abord présenté le formalisme «classique» des MDP, où l'espace d'états est entièrement énuméré. Les MDP sont généralement optimisés par des algorithmes de programmation dynamique, tels les algorithmes d'itération de la valeur ou d'itération de la politique, dont la complexité est polynomiale en le nombre d'actions et d'états. Aussi, l'optimisation des MDP est d'autant plus difficile que l'espace d'états est grand, le nombre d'actions étant souvent négligeable devant le nombre d'états.

Ainsi, des techniques de décomposition de MDP ont été proposées afin d'optimiser, de manière asynchrone, des sous-MDP définis dans des petites partitions de l'espace d'états. Ces techniques sont particulièrement efficaces lorsque l'espace d'états est constitué de régions faiblement communiquantes, comme dans certains problèmes de navigation. Néanmoins, selon le principe de «*curse of dimensionality*» de Bellman [71], la taille de l'espace d'états augmente exponentiellement avec le nombre de caractéristiques qui le décrivent. Si ce constat est sans effet sur un espace d'états purement géographique par exemple, il est en revanche catastrophique pour des problèmes de navigation et de prise d'information, qui combinent à la fois des caractéristiques de déplacement et d'objectifs à atteindre. Ainsi, les techniques de décomposition de MDP ne s'appliquent plus, car l'espace d'états ne peut même plus être construit.

Heureusement, il existe des modèles factorisés de MDP, où l'espace d'états est décrit en intention par des caractéristiques qui ne sont instanciées qu'au besoin. Des algorithmes symboliques d'itération de la valeur et d'itération de la politique ont été proposés, tel `SPUDD`, qui explorent et évaluent les états par lots sans les expliciter, grâce à une instanciation partielle de leurs caractéristiques. Ces algorithmes reposent sur un encodage symbolique et fonctionnel des données du MDP, à l'aide de Diagrammes de Décision Binaires ou Algébriques.

Les MDP factorisés permettent de modéliser des problèmes de planification stochastique complexes, que des modèles énumérés ne parviennent pas à construire. Cependant, leur optimisation mène à explorer des sous-espaces d'états de plus en plus petits et nombreux au fur et à mesure de l'amélioration de la stratégie, si bien qu'elle nécessite souvent d'explorer quasiment tout l'espace d'états. Ainsi, l'optimisation des problèmes de grande taille reste problématique, voire impossible dans certains cas.

Par conséquent, nous avons généralisé les techniques de décomposition de MDP énumérés aux MDP factorisés, afin de réduire la taille de leur espace d'états. A cette fin, nous avons proposé un modèle décomposé et factorisé de MDP, où l'espace d'états est décrit par des caractéristiques dont certaines sont réduites en macro-caractéristiques abstraites d'arité réduite. La taille de l'espace d'états total, qui est exponentielle en l'arité des caractéristiques, est donc réduite, ce qui facilite l'optimisation du MDP. L'arité des caractéristiques est réduite en décomposant des sous-MDP énumérés sur les sous-espaces d'états engendrés par ces caractéristiques. Nous avons également établi et démontré le théorème de reconstruction du MDP factorisé abstrait, contenant les caractéristiques réduites. Sur la base de ce théorème, nous avons proposé un algorithme intitulé `SVAR` («State Variable Automatic Reduction») qui vise à réduire automatiquement l'arité de caractéristiques données.

Néanmoins, ces recherches ont été menées récemment en complément de nos autres travaux, si bien que nous n'avons pas eu le temps de les implémenter afin de tester leur efficacité. Aussi, nous avons calculé le plus précisément possible les complexités dans le pire cas de notre algorithme `SVAR` dans le but de prévoir son comportement en pratique. Il semblerait que le gain en temps de calcul et en mémoire pour l'optimisation du MDP réduit soit nettement supérieure au coût de la réduction, à condition que les caractéristiques réduites et les récompenses dépendent chacune de peu de caractéristiques.

De plus, la décomposition des sous-MDP énumérés induit dans le cas général une perte d'optimalité, qui est d'autant plus petite que les variations dynamiques des caractéristiques réduites sont importantes devant celles des caractéristiques dont elles dépendent. Cette perte d'optimalité est malheureusement difficile à établir en théorie, si bien que seul un échantillon statistique de tests permettraient de l'évaluer. Toutefois, dans le cas des problèmes de déplacement et de prise d'information, la

caractéristique de navigation varie beaucoup plus fréquemment que les autres caractéristiques et son arité est souvent très grande, si bien que notre algorithme `SVAR` *devrait* améliorer sensiblement l'optimisation de ce type de problèmes.

Cependant, si l'algorithme `SVAR` vise à faciliter l'optimisation du MDP après sa réduction automatique, il ne consiste pas à simplifier la phase de modélisation du MDP. En effet, la définition manuelle des transitions symboliques des MDP factorisés est pénible et laborieuse lorsque certaines caractéristiques ont une très grande arité, car elle nécessite alors de modéliser des arbres de décision de largeur très importante.

Par conséquent, nous proposons de modéliser le MDP factorisé directement à partir de caractéristiques hiérarchiques de faible arité, qui sont des abstractions de caractéristiques concrètes de grande arité, et dont les valeurs sont des partitions de l'ensemble des valeurs des caractéristiques concrètes. Dans le cas d'une mission de déplacement et de prise d'information, la caractéristique de navigation pourrait par exemple être définie au niveau abstrait des régions, plutôt qu'au niveau microscopique des positions élémentaires.

Néanmoins, la réduction de l'espace d'états dans le modèle nécessite de définir des macro-actions compatibles avec le niveau d'abstraction des caractéristiques réduites. Par exemple, les actions de déplacement élémentaires ne permettent pas de transiter entre les régions abstraites du sous-espace de navigation. Aussi, le modèle d'action doit être défini au niveau abstrait des partitions de l'ensemble des valeurs des caractéristiques réduites, de telle sorte que chaque macro-action agisse sur une partition spécifique de la caractéristique abstraite.

Les macro-actions peuvent être générées à l'aide de techniques de décomposition, appliquées au sous-MDP énuméré défini dans le sous-espace de la caractéristique de grande arité. Ces techniques ont l'avantage de produire, dans chaque partition, des macro-actions dont la dynamique microscopique est optimale au regard des récompenses extérieures à la partition. Cependant, elles génèrent souvent un grand nombre de macro-actions, au point que la modélisation du MDP factorisé hiérarchique à l'aide de ces macro-actions peut devenir rédhibitoire.

Toutefois, la dynamique du système décisionnel étant supposée par essence microscopique, les effets de chaque macro-action ne dépendent a priori que de la partition de la macro-caractéristique où elle s'applique. Si une macro-action dépendait d'une valeur spécifique de la macro-caractéristique, cela signifierait que le modèle est initialement défini à un niveau macroscopique. Par conséquent, nous avons proposé un modèle hiérarchique et générique de MDP factorisé, paramétré par une macro-action variable, et donc aussi par la partition de la caractéristique réduite où elle s'applique. Nous avons également proposé un algorithme qui vise à instancier automatiquement toutes les macro-actions du MDP hiérarchique, connaissant un ensemble de macro-actions disponibles, et sur la base d'un modèle générique et abstrait de la dynamique du système décisionnel.

Nous avons implémenté et testé notre approche générique et hiérarchique sur des problèmes de déplacement et de prise d'information de type ReSSAC. Le sous-espace engendré par la caractéristique de navigation (position géographique) a été partitionné en régions faiblement communiquantes, à l'aide de techniques de décomposition de MDP énumérés. Nous avons pu ensuite définir un modèle simple, compact et générique de mission, dont le MDP factorisé est automatiquement instancié, préalablement à l'optimisation de la stratégie. Alors que des modèles «classiques» et non structurés de MDP, où tout l'espace d'états est énuméré, ne permettent pas de construire en temps raisonnable (moins d'une heure) des missions de petites tailles (moins de 10^5 états), nous sommes parvenu, grâce à notre modèle générique et hiérarchique, à instancier des missions de tailles quelconques en un temps négligeable devant le temps d'optimisation de la stratégie.

Toutefois, nous n'avons pas comparé notre modèle générique et abstrait à un modèle factorisé équivalent, où les caractéristiques de grande arité ne seraient pas réduites. Bien que cette comparaison reste à faire, les décompositions des sous-MDP énumérés, définis dans les partitions des caractéristiques abstraites, *devraient* accélérer le calcul de la stratégie optimale : les macro-actions produites constituent en effet une optimisation partielle de la stratégie globale, au niveau des sous-espaces engendrés par les caractéristiques réduites.

Cependant, notre modèle ne s'applique plus lorsque les effets de certaines macro-actions dépendent de partitions spécifiques des caractéristiques réduites, indépendamment de la partition où chaque macro-action agit. Considérons par exemple une mission de type ReSSAC telle que la dynamique de déplacement est microscopique, mais telle que les communications avec la station sol ne sont possibles que dans certaines régions macroscopiques. La définition de la mission étant par essence microscopique, notre modèle hiérarchique semble s'appliquer. Néanmoins, avec le modèle actuel, il est impossible de spécifier que certaines actions, comme les communications radio par exemple, dépendent de régions statiques et indépendantes de celles où les actions s'appliquent. Il serait donc intéressant d'enrichir la syntaxe de notre modèle hiérarchique et générique, de sorte à autoriser des dépendances statiques vis-à-vis de certaines partitions données de la caractéristique réduite.

Toutefois, les tests de notre modèle hiérarchique ont montré qu'un algorithme d'optimisation globale des MDP factorisés, tel `SPUDD`, est incapable d'optimiser en temps raisonnable (moins d'une heure) des stratégies sur des espaces de plus de 10^8 états. Or, lorsque des états initiaux possibles de la mission sont connus, et que les récompenses peuvent être identifiées, il est possible de réduire l'exploration des états visités par la stratégie optimale à un sous-espace atteignable de taille raisonnable.

Aussi, nous avons proposé un algorithme de programmation dynamique symbolique et focalisé, appelé `sfDP` («Symbolic Focused Dynamic Programming»), qui cible la recherche des états atteignables par la stratégie optimale sur les récompenses du

MDP. Comme sLAO* dont il est inspiré, notre algorithme utilise une heuristique calculée préalablement à l'optimisation du MDP, qui guide l'expansion du sous-espace atteignable vers les récompenses du problème. Cependant, alors que l'heuristique de sLAO* est une approximation globale de la fonction de valeur qui nécessite d'explorer initialement tout l'espace d'états, notre heuristique consiste en une politique qui maximise la probabilité d'atteindre les récompenses du MDP, optimisée sur un sous-espace atteignable initial qui contient toutes les récompenses.

Contrairement à sLAO*, sfDP connaît les positions des récompenses dans l'espace d'états, ce qui nous a permis d'implémenter deux modes de fonctionnement différents : un mode opportuniste, si une partie seulement des récompenses est imposée au planificateur sans l'empêcher toutefois d'en récolter d'autres, et un mode optimal, si toutes les récompenses du MDP doivent être atteintes. Ainsi, en mode opportuniste, le sous-espace atteignable calculé par notre heuristique ne contient pas toutes les récompenses du problème, au point que l'optimisation de la stratégie est ciblée sur les récompenses internes au sous-espace atteignable. Les récompenses extérieures sont découvertes de manière opportuniste lors de l'expansion des états atteignables.

Nous avons testé sfDP sur des missions de déplacement et de prise d'information de tailles différentes, qui nous ont permis d'identifier les meilleures conditions d'arrêt de l'expansion du sous-espace atteignable, ainsi que la meilleure façon d'explorer de nouveaux états lors de cette expansion. Ainsi, il suffit d'arrêt l'expansion des états atteignables lorsqu'au moins un état vérifiant toutes les condition de buts est atteint ; les autres états satisfaisant également toutes les conditions de buts ne servent pas à améliorer la stratégie optimisée. De plus, l'expansion du sous-espace atteignable en appliquant toutes les actions possibles depuis les états atteignables courants permet certes de découvrir opportunément un peu plus de récompenses non imposées qu'en suivant la politique courante, mais au détriment du temps d'optimisation.

De plus, nous avons proposé et implémenté une version en-ligne incrémentale de sfDP, appelée IsfDP («Incremental sfDP»), où les récompenses de la mission sont imposées incrémentalement à l'agent lors du déroulement de la mission. Ainsi, plutôt que de replanifier toute la stratégie depuis les états initiaux de la mission à chaque nouvelle requête de sous-buts, IsfDP permet d'améliorer la stratégie courante sur la base de celle optimisée avec les précédentes requêtes de sous-buts. Les nouveaux états atteignables sont développés en partant du sous-espace atteignable précédent, et non depuis les états initiaux de la mission. Ainsi, suivant la topologie du problème et le placement des récompenses, la nouvelle stratégie peut être mise à jour sur un ensemble d'états de taille quasiment constante. Aussi, nous avons pu constater sur des missions de déplacement et de prise d'information où les récompenses sont à peu près équidistantes, que le temps d'amélioration de la stratégie est quasiment constant entre deux requêtes de sous-buts.

Enfin, nous avons participé en 2004 à la Première Compétition Internationale de

Planification Stochastique, avec une version malheureusement immature et boguée de `sfDP`, qui nous a toutefois permis de finir en troisième position derrière `sLAO`* et `NMRDPP` dans la catégorie MDP. Ces résultats nous encouragent à participer à la deuxième compétition qui aura lieu en 2006, à l'aide de la version actuelle de `sfDP`, dont l'heuristique et le mécanisme d'expansion du sous-espace atteignable ont été grandement améliorés depuis la précédente compétition.

Cinquième partie

Annexes

Annexe A

Architecture de l'outil de modélisation et résolution de missions ReSSAc

A.1 Architecture générale

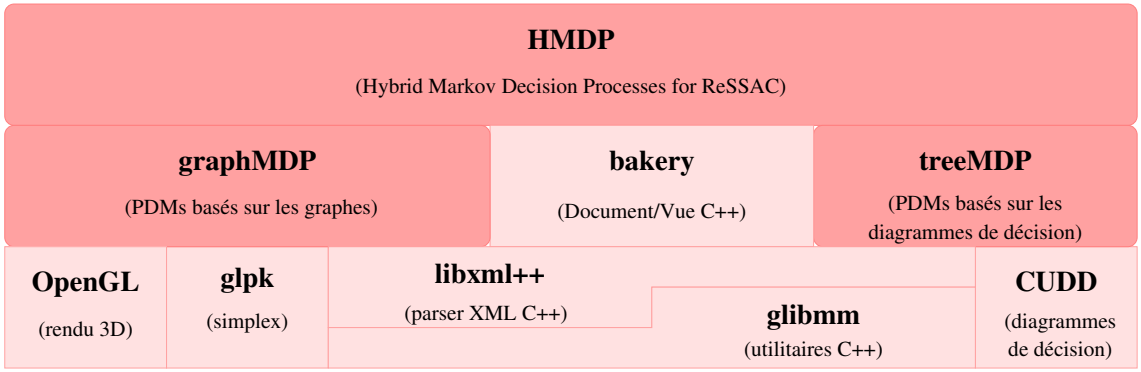


FIG. A.1 – Architecture logicielle de l'outil de modélisation HMDP

A.2 DTD d'une mission ReSSAC

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
DTD for HMDP.
-->
```

```
<!ELEMENT hmdp_xml_doc (dtm,waypoints,regions,state_variables,
                        probability_trees,reward_tree,preferences)>

<!ELEMENT dtm EMPTY>
<!ATTLIST dtm
file CDATA #REQUIRED
texture CDATA #REQUIRED
>

<!ELEMENT waypoints (waypoint+)>
<!ELEMENT waypoint (action_transitions+)>
<!ATTLIST waypoint
label CDATA #REQUIRED
is_sub_goal (yes|no) #REQUIRED
x_coordinate CDATA #REQUIRED
y_coordinate CDATA #REQUIRED
z_coordinate CDATA #REQUIRED
>
<!ELEMENT action_transitions (edge+)>
<!ATTLIST action_transitions
label CDATA #REQUIRED
>
<!ELEMENT edge EMPTY >
<!ATTLIST edge
reachable_waypoint CDATA #REQUIRED
probability CDATA #REQUIRED
reward CDATA #REQUIRED
>

<!ELEMENT regions (region+)>
<!ELEMENT region (control_point+)>
<!ATTLIST region
label CDATA #REQUIRED
z_min CDATA #REQUIRED
z_max CDATA #REQUIRED
>
<!ELEMENT control_point EMPTY>
<!ATTLIST control_point
x_coordinate CDATA #REQUIRED
y_coordinate CDATA #REQUIRED
```

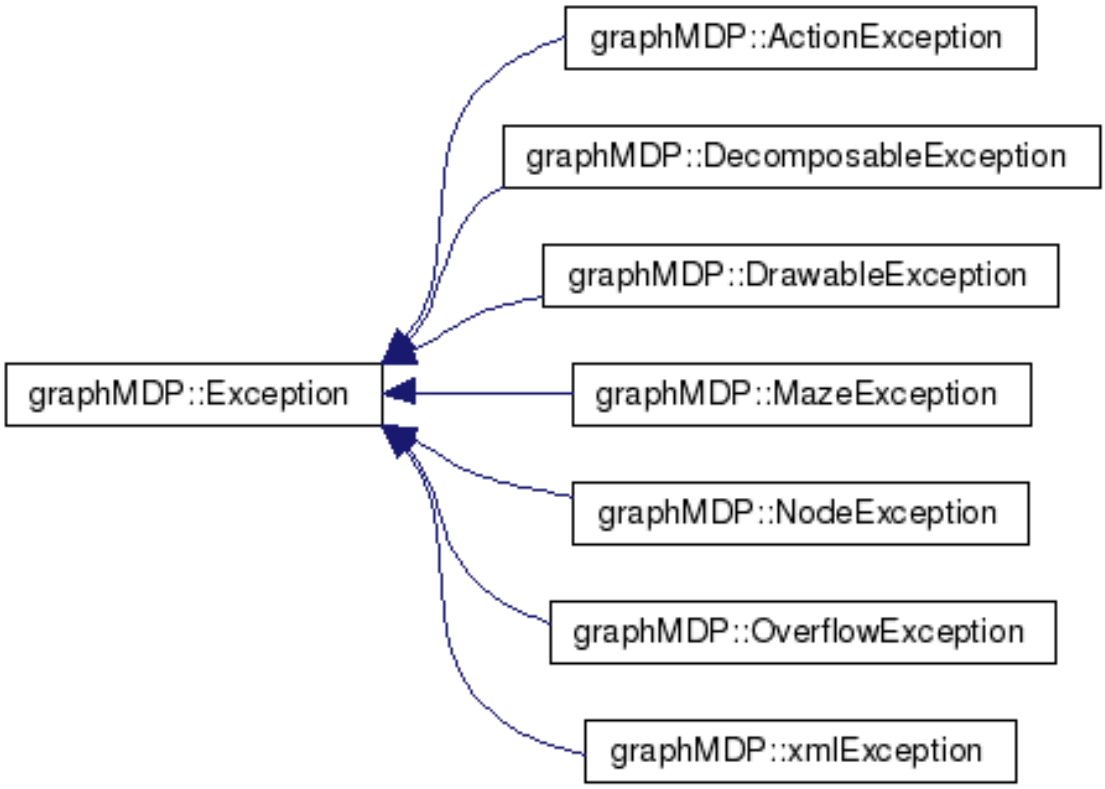
```
>

<!ELEMENT state_variables (state_variable+)>
<!ELEMENT state_variable EMPTY>
<!ATTLIST state_variable
label CDATA #REQUIRED
region_dependence (yes|no) #REQUIRED
values_labels CDATA #REQUIRED
>
<!ELEMENT probability_trees (probability_tree+)>
<!ELEMENT probability_tree (node|leaf)>
<!ATTLIST probability_tree
state_variable CDATA #REQUIRED
>
<!ELEMENT reward_tree (node|leaf)>
<!ELEMENT node ((node|leaf)+)>
<!ATTLIST node
state_variable CDATA #REQUIRED
pre_action (yes|no) #REQUIRED
>
<!ELEMENT leaf EMPTY >
<!ATTLIST leaf
value CDATA #REQUIRED
>

<!ELEMENT preferences (waypoints_preferences,graph_view_preferences)>

<!ELEMENT waypoints_preferences EMPTY>
<!ATTLIST waypoints_preferences
radius CDATA #REQUIRED
>

<!ELEMENT graph_view_preferences (show_objects,showing_transitions)>

<!ELEMENT show_objects EMPTY>
<!ATTLIST show_objects
waypoints (yes|no) #REQUIRED
transitions (yes|no) #REQUIRED
regions (yes|no) #REQUIRED
>
```

```
<!ELEMENT showing_transitions EMPTY>
<!ATTLIST showing_transitions
mode (all_states_all_actions|all_states_single_action|
      single_state_all_actions|single_state_single_action) #REQUIRED
>
```

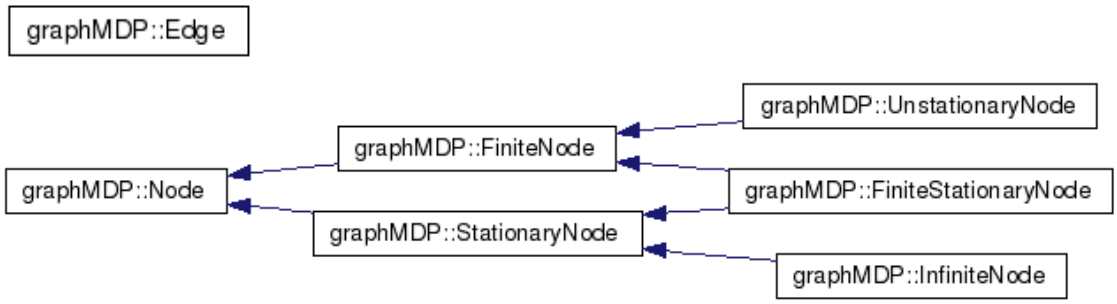
Annexe B

Architecture de la bibliothèque graphMDP

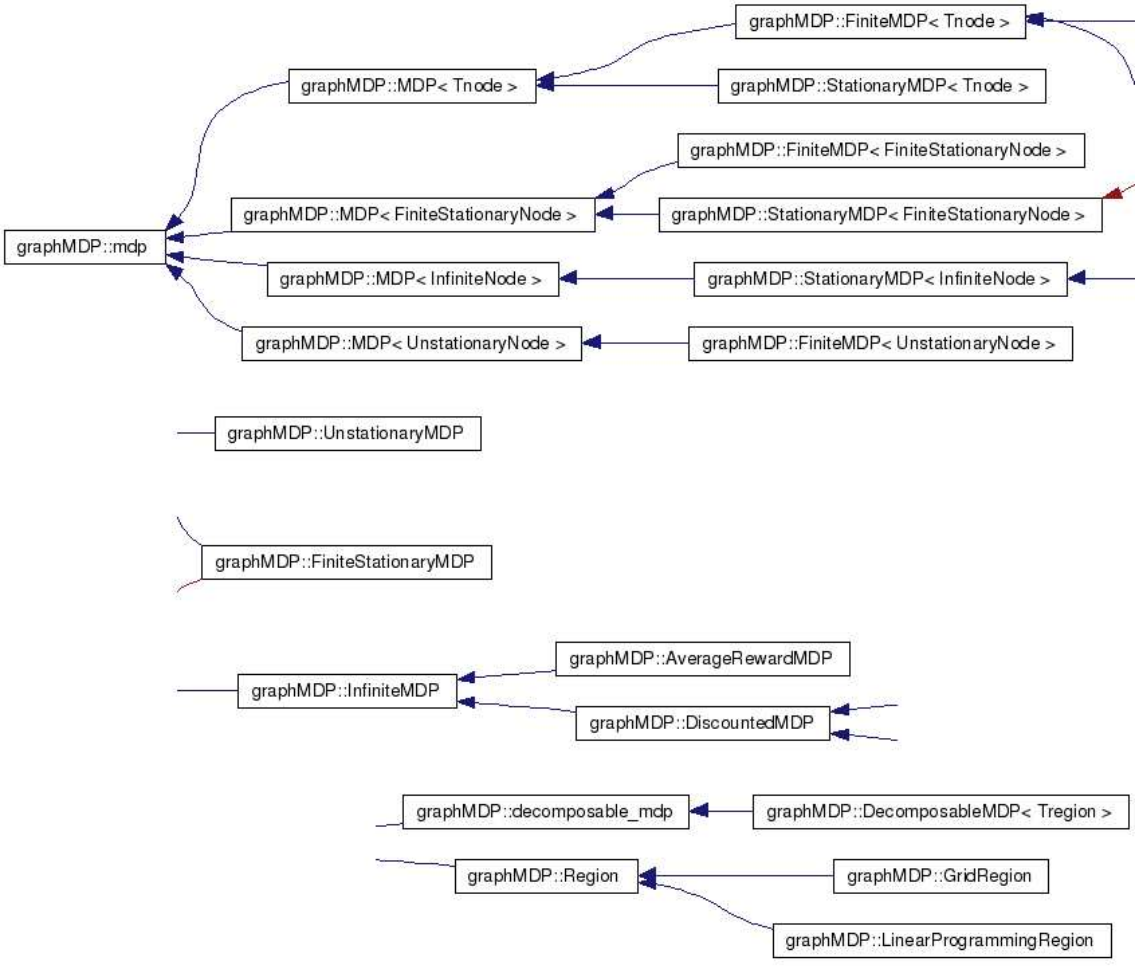
B.1 Exceptions



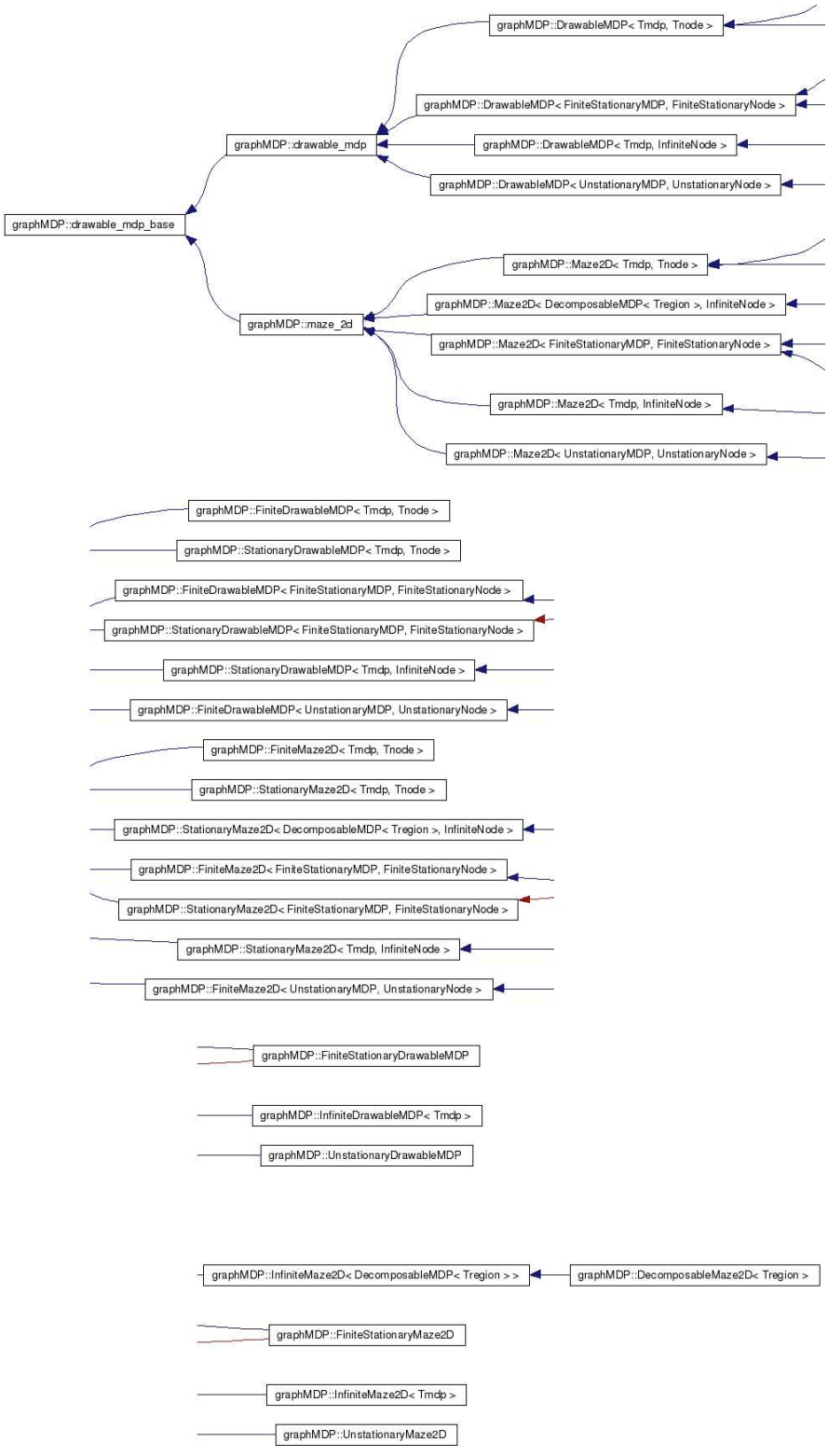
B.2 Graphe



B.3 Critères d'optimisation



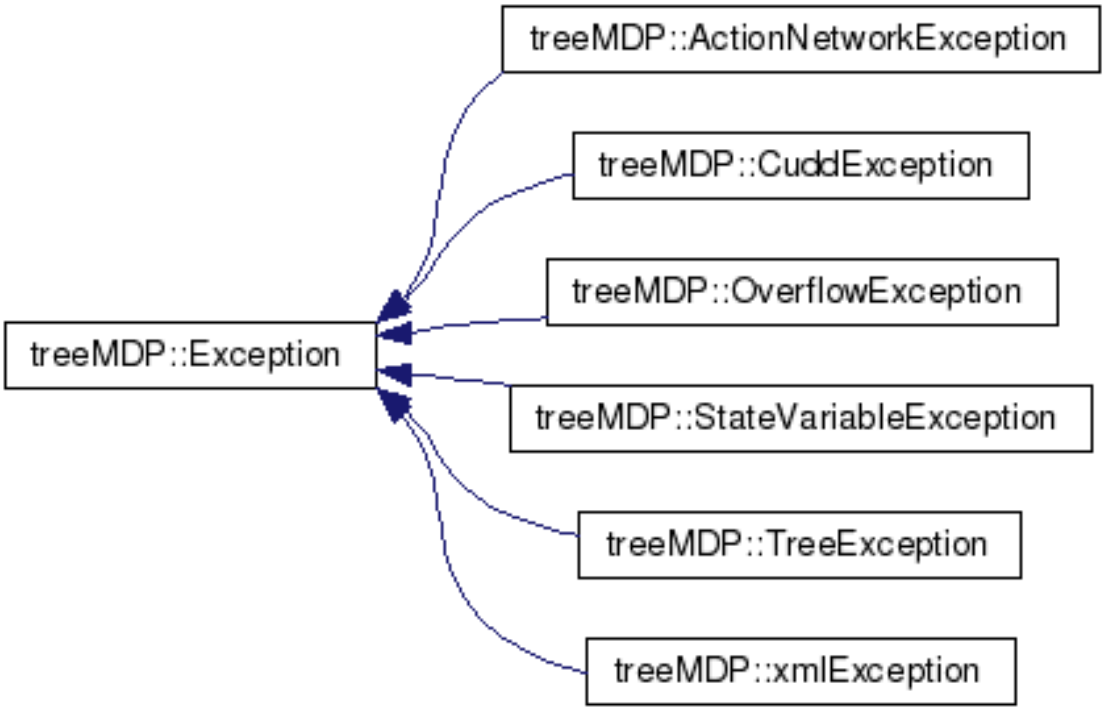
B.4 Représentation 3D



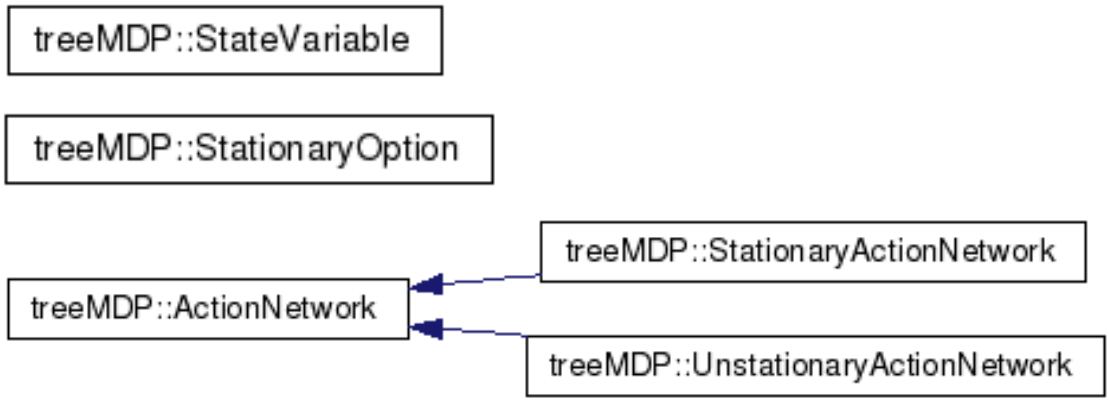
Annexe C

Architecture de la la bibliothèque treeMDP

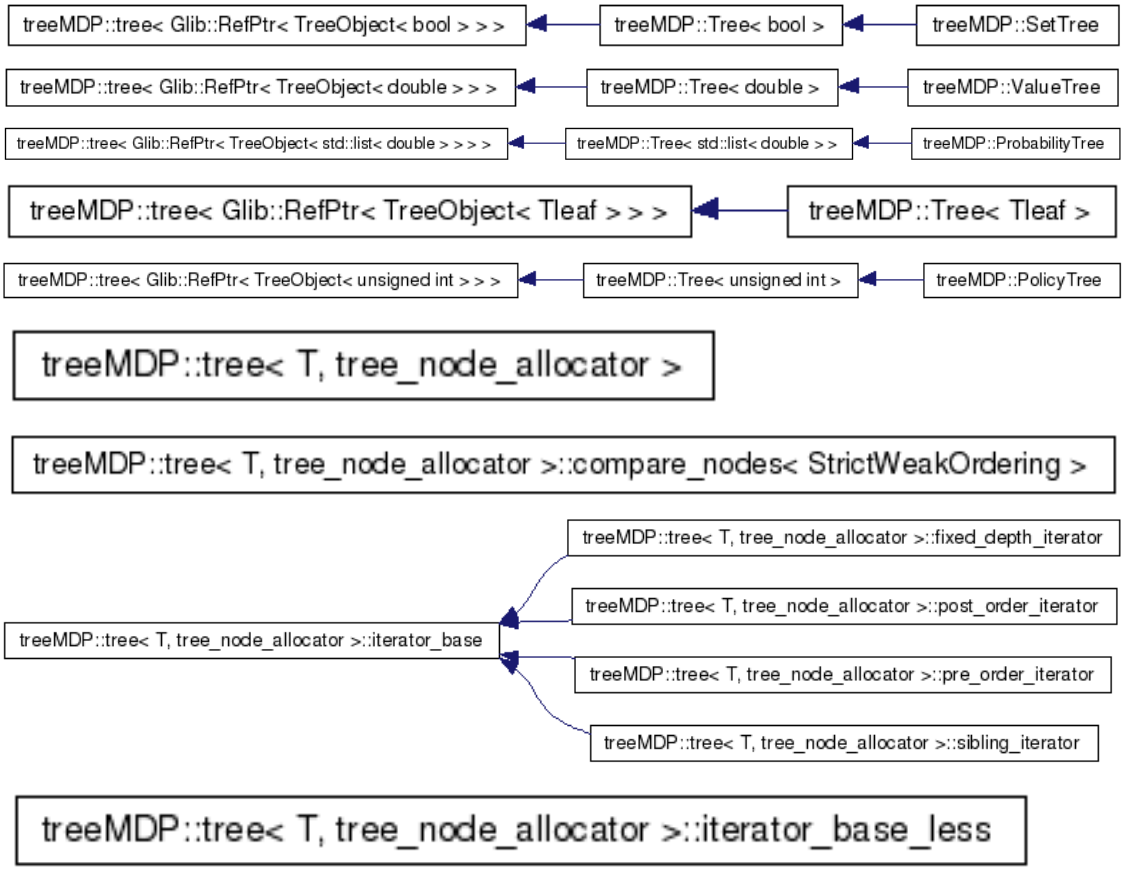
C.1 Exceptions

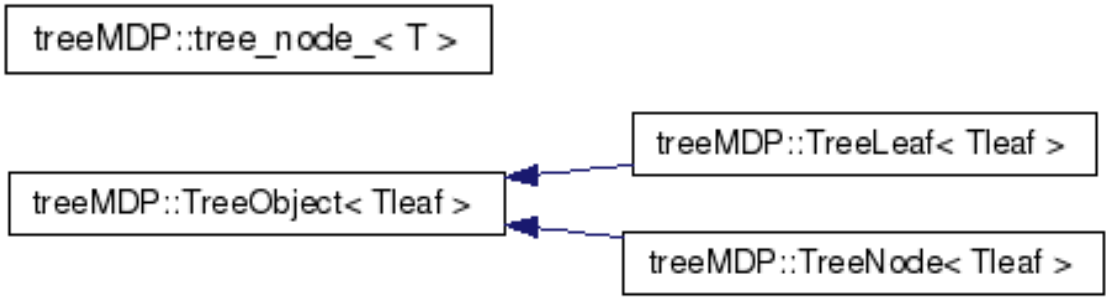


C.2 Variables d'état et actions

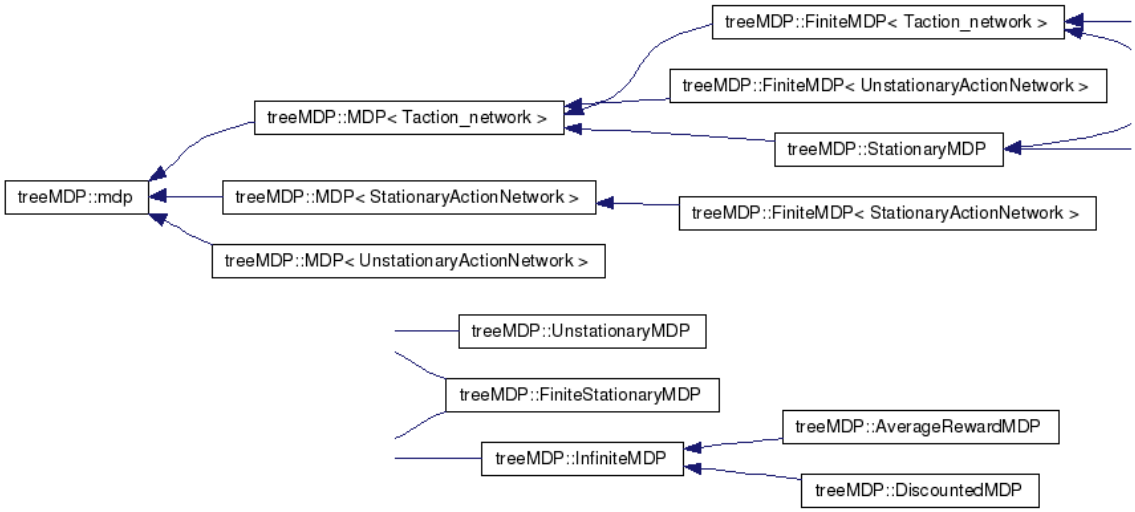


C.3 Arbres de décision





C.4 Critères d'optimisation



Annexe D

Domaines de la Première Compétition Internationale de Planification Stochastique

D.1 bw-c-pc-8

```
(define (domain bw-c-pc-8)
 (:requirements :adl :probabilistic-effects :fluents :rewards)
 (:types block table)
 (:constants table - table)
 (:predicates
  (is-red ?block - block)
  (is-blue ?block - block)
  (is-green ?block - block)
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
                     (on-top-of ?top ?bottom)
                     (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (decrease (reward) 1)
              (probabilistic 0.75 (and (holding ?top)
                                       (not (on-top-of ?top ?bottom)))
                                  0.25 (when (not (= ?bottom table))
                                             (and (not (on-top-of ?top ?bottom))
                                                  (on-top-of ?top table)))))
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
```

```
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
  :effect                (and (not (holding ?top))
               (probabilistic 0.75 (on-top-of ?top ?bottom)
                              0.25 (on-top-of ?top table)))
 )
)
(define (problem bw-c-pc-8)
 (:domain bw-c-pc-8)
 (:objects block0 block1 block2 block3 block4 block5 block6 block7 - block)
 (:init
  (on-top-of block0 block1)
  (on-top-of block1 table)
  (on-top-of block2 table)
  (on-top-of block3 block4)
  (on-top-of block4 block5)
  (on-top-of block5 table)
  (on-top-of block6 table)
  (on-top-of block7 table)
  (is-red block0)
  (is-red block1)
  (is-green block2)
  (is-red block3)
  (is-green block4)
  (is-red block5)
  (is-green block6)
  (is-blue block7)
 )
 (:goal
  (and
   (exists (?fb0 - block)
    (and
     (is-green ?fb0)
   (exists (?fb1 - block)
    (and
     (not (= ?fb0 ?fb1))
     (is-green ?fb1)
     (on-top-of ?fb0 ?fb1)
   (exists (?fb2 - block)
    (and
     (not (= ?fb0 ?fb2))
     (not (= ?fb1 ?fb2))
     (is-blue ?fb2)
     (on-top-of ?fb1 ?fb2)
   (exists (?fb3 - block)
    (and
     (not (= ?fb0 ?fb3))
```

```
     (not (= ?fb1 ?fb3))
     (not (= ?fb2 ?fb3))
     (is-red ?fb3)
     (on-top-of ?fb2 ?fb3)
   (exists (?fb4 - block)
    (and
     (not (= ?fb0 ?fb4))
     (not (= ?fb1 ?fb4))
     (not (= ?fb2 ?fb4))
     (not (= ?fb3 ?fb4))
     (is-red ?fb4)
     (on-top-of ?fb3 table)
   (exists (?fb5 - block)
    (and
     (not (= ?fb0 ?fb5))
     (not (= ?fb1 ?fb5))
     (not (= ?fb2 ?fb5))
     (not (= ?fb3 ?fb5))
     (not (= ?fb4 ?fb5))
     (is-red ?fb5)
     (on-top-of ?fb4 ?fb5)
   (exists (?fb6 - block)
    (and
     (not (= ?fb0 ?fb6))
     (not (= ?fb1 ?fb6))
     (not (= ?fb2 ?fb6))
     (not (= ?fb3 ?fb6))
     (not (= ?fb4 ?fb6))
     (not (= ?fb5 ?fb6))
     (is-green ?fb6)
     (on-top-of ?fb5 ?fb6)
   (exists (?fb7 - block)
    (and
     (not (= ?fb0 ?fb7))
     (not (= ?fb1 ?fb7))
     (not (= ?fb2 ?fb7))
     (not (= ?fb3 ?fb7))
     (not (= ?fb4 ?fb7))
     (not (= ?fb5 ?fb7))
     (not (= ?fb6 ?fb7))
     (is-red ?fb7)
     (on-top-of ?fb6 ?fb7)
     (on-top-of ?fb7 table)
))))))))))))))
  )
 )
 (:goal-reward 500)
)
```

D.2 bw-c-pc-nr-8

```
(define (domain bw-c-pc-nr-nr-8)
 (:requirements :adl :probabilistic-effects :fluents)
 (:types block table)
 (:constants table - table)
 (:predicates
  (is-red ?block - block)
  (is-blue ?block - block)
  (is-green ?block - block)
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
                     (on-top-of ?top ?bottom)
                     (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (probabilistic 0.75 (and (holding ?top)
                                                       (not (on-top-of ?top ?bottom)))
                                  0.25 (when (not (= ?bottom table))
                                             (and (not (on-top-of ?top ?bottom))
                                                  (on-top-of ?top table)))))
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
  :effect                 (and (not (holding ?top))
               (probabilistic 0.75 (on-top-of ?top ?bottom)
                                   0.25 (on-top-of ?top table)))
 )
)
(define (problem bw-c-pc-nr-8)
 (:domain bw-c-pc-nr-nr-8)
 (:objects block0 block1 block2 block3 block4 block5 block6 block7 - block)
 (:init
  (on-top-of block0 block1)
  (on-top-of block1 table)
  (on-top-of block2 table)
  (on-top-of block3 block4)
  (on-top-of block4 block5)
  (on-top-of block5 table)
  (on-top-of block6 table)
  (on-top-of block7 table)
  (is-red block0)
```

```
 (is-red block1)
 (is-green block2)
 (is-red block3)
 (is-green block4)
 (is-red block5)
 (is-green block6)
 (is-blue block7)
)
(:goal
 (and
  (exists (?fb0 - block)
   (and
    (is-green ?fb0)
  (exists (?fb1 - block)
   (and
    (not (= ?fb0 ?fb1))
    (is-green ?fb1)
    (on-top-of ?fb0 ?fb1)
  (exists (?fb2 - block)
   (and
    (not (= ?fb0 ?fb2))
    (not (= ?fb1 ?fb2))
    (is-blue ?fb2)
    (on-top-of ?fb1 ?fb2)
  (exists (?fb3 - block)
   (and
    (not (= ?fb0 ?fb3))
    (not (= ?fb1 ?fb3))
    (not (= ?fb2 ?fb3))
    (is-red ?fb3)
    (on-top-of ?fb2 ?fb3)
  (exists (?fb4 - block)
   (and
    (not (= ?fb0 ?fb4))
    (not (= ?fb1 ?fb4))
    (not (= ?fb2 ?fb4))
    (not (= ?fb3 ?fb4))
    (is-red ?fb4)
    (on-top-of ?fb3 table)
  (exists (?fb5 - block)
   (and
    (not (= ?fb0 ?fb5))
    (not (= ?fb1 ?fb5))
    (not (= ?fb2 ?fb5))
    (not (= ?fb3 ?fb5))
    (not (= ?fb4 ?fb5))
    (is-red ?fb5)
    (on-top-of ?fb4 ?fb5)
  (exists (?fb6 - block)
```

```
    (and
     (not (= ?fb0 ?fb6))
     (not (= ?fb1 ?fb6))
     (not (= ?fb2 ?fb6))
     (not (= ?fb3 ?fb6))
     (not (= ?fb4 ?fb6))
     (not (= ?fb5 ?fb6))
     (is-green ?fb6)
     (on-top-of ?fb5 ?fb6)
   (exists (?fb7 - block)
    (and
     (not (= ?fb0 ?fb7))
     (not (= ?fb1 ?fb7))
     (not (= ?fb2 ?fb7))
     (not (= ?fb3 ?fb7))
     (not (= ?fb4 ?fb7))
     (not (= ?fb5 ?fb7))
     (not (= ?fb6 ?fb7))
     (is-red ?fb7)
     (on-top-of ?fb6 ?fb7)
     (on-top-of ?fb7 table)
))))))))))))))))
  )
 )
)
```

D.3 bw-nc-pc-11

```
(define (domain bw-nc-pc-11)
 (:requirements :adl :probabilistic-effects :fluents :rewards)
 (:types block table)
 (:constants table - table)
 (:predicates
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
                     (on-top-of ?top ?bottom)
                     (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (decrease (reward) 1)
              (probabilistic 0.75 (and (holding ?top)
                                       (not (on-top-of ?top ?bottom)))
                             0.25 (when (not (= ?bottom table))
                                        (and (not (on-top-of ?top ?bottom))
                                             (on-top-of ?top table)))))
```

```
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
  :effect                 (and (not (holding ?top))
              (probabilistic 0.75 (on-top-of ?top ?bottom)
                             0.25 (on-top-of ?top table)))
 )
)
(define (problem bw-nc-pc-11)
 (:domain bw-nc-pc-11)
 (:objects block0 block1 block2 block3 block4 block5 block6 block7
                                          block8 block9 block10 - block)
 (:init
  (on-top-of block0 block1)
  (on-top-of block1 block2)
  (on-top-of block2 table)
  (on-top-of block3 block4)
  (on-top-of block4 table)
  (on-top-of block5 block6)
  (on-top-of block6 block7)
  (on-top-of block7 block8)
  (on-top-of block8 table)
  (on-top-of block9 table)
  (on-top-of block10 table)
 )
 (:goal
  (and
   (on-top-of block0 block6)
   (on-top-of block6 block4)
   (on-top-of block4 block10)
   (on-top-of block10 block2)
   (on-top-of block2 block9)
   (on-top-of block9 table)
   (on-top-of block1 block8)
   (on-top-of block8 block3)
   (on-top-of block3 block7)
   (on-top-of block7 block5)
   (on-top-of block5 table)

  )
 )
 (:goal-reward 500)
)
```

D.4 bw-nc-pc-15

```
(define (domain bw-nc-pc-15)
 (:requirements :adl :probabilistic-effects :fluents :rewards)
 (:types block table)
 (:constants table - table)
 (:predicates
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
                     (on-top-of ?top ?bottom)
                     (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (decrease (reward) 1)
              (probabilistic 0.75 (and (holding ?top)
                                       (not (on-top-of ?top ?bottom)))
                                  0.25 (when (not (= ?bottom table))
                                             (and (not (on-top-of ?top ?bottom))
                                                  (on-top-of ?top table)))))
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
  :effect                 (and (not (holding ?top))
              (probabilistic 0.75 (on-top-of ?top ?bottom)
                                  0.25 (on-top-of ?top table)))
 )
)
(define (problem bw-nc-pc-15)
 (:domain bw-nc-pc-15)
 (:objects block0 block1 block2 block3 block4 block5 block6 block7
       block8 block9 block10 block11 block12 block13 block14 - block)
 (:init
  (on-top-of block0 block1)
  (on-top-of block1 block2)
  (on-top-of block2 block3)
  (on-top-of block3 table)
  (on-top-of block4 block5)
  (on-top-of block5 block6)
  (on-top-of block6 table)
  (on-top-of block7 block8)
  (on-top-of block8 block9)
  (on-top-of block9 block10)
```

```
  (on-top-of block10 block11)
  (on-top-of block11 block12)
  (on-top-of block12 table)
  (on-top-of block13 table)
  (on-top-of block14 table)
 )
 (:goal
  (and
   (on-top-of block0 block6)
   (on-top-of block6 block4)
   (on-top-of block4 block10)
   (on-top-of block10 block2)
   (on-top-of block2 block13)
   (on-top-of block13 block9)
   (on-top-of block9 block11)
   (on-top-of block11 block1)
   (on-top-of block1 block8)
   (on-top-of block8 block14)
   (on-top-of block14 block3)
   (on-top-of block3 block7)
   (on-top-of block7 block5)
   (on-top-of block5 table)
   (on-top-of block12 table)

  )
 )
 (:goal-reward 500)
)
```

D.5 bw-nc-pc-18

```
(define (domain bw-nc-pc-18)
 (:requirements :adl :probabilistic-effects :fluents :rewards)
 (:types block table)
 (:constants table - table)
 (:predicates
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
                     (on-top-of ?top ?bottom)
                     (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (decrease (reward) 1)
              (probabilistic 0.75 (and (holding ?top)
                                       (not (on-top-of ?top ?bottom)))
```

```
                              0.25 (when (not (= ?bottom table))
                                         (and (not (on-top-of ?top ?bottom))
                                              (on-top-of ?top table)))))
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
  :effect                 (and (not (holding ?top))
               (probabilistic 0.75 (on-top-of ?top ?bottom)
                              0.25 (on-top-of ?top table)))
 )
)
(define (problem bw-nc-pc-18)
 (:domain bw-nc-pc-18)
 (:objects block0 block1 block2 block3 block4 block5 block6 block7 block8
 block9 block10 block11 block12 block13 block14 block15 block16 block17 - block)
 (:init
  (on-top-of block0 block1)
  (on-top-of block1 block2)
  (on-top-of block2 block3)
  (on-top-of block3 table)
  (on-top-of block4 block5)
  (on-top-of block5 block6)
  (on-top-of block6 block7)
  (on-top-of block7 table)
  (on-top-of block8 block9)
  (on-top-of block9 block10)
  (on-top-of block10 block11)
  (on-top-of block11 block12)
  (on-top-of block12 block13)
  (on-top-of block13 block14)
  (on-top-of block14 table)
  (on-top-of block15 table)
  (on-top-of block16 table)
  (on-top-of block17 table)
 )
 (:goal
  (and
   (on-top-of block0 block5)
   (on-top-of block5 block3)
   (on-top-of block3 block17)
   (on-top-of block17 block9)
   (on-top-of block9 block12)
   (on-top-of block12 block1)
   (on-top-of block1 block10)
   (on-top-of block10 block8)
```

```
   (on-top-of block8 block15)
   (on-top-of block15 block7)
   (on-top-of block7 block13)
   (on-top-of block13 block2)
   (on-top-of block2 block6)
   (on-top-of block6 block4)
   (on-top-of block4 block14)
   (on-top-of block14 table)
   (on-top-of block11 table)
   (on-top-of block16 table)

  )
 )
 (:goal-reward 500)
)
```

D.6 bw-nc-pc-21

```
(define (domain bw-nc-pc-21)
 (:requirements :adl :probabilistic-effects :fluents :rewards)
 (:types block table)
 (:constants table - table)
 (:predicates
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
                     (on-top-of ?top ?bottom)
                     (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (decrease (reward) 1)
              (probabilistic 0.75 (and (holding ?top)
                                       (not (on-top-of ?top ?bottom)))
                                  0.25 (when (not (= ?bottom table))
                                             (and (not (on-top-of ?top ?bottom))
                                                  (on-top-of ?top table)))))
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
  :effect                 (and (not (holding ?top))
              (probabilistic 0.75 (on-top-of ?top ?bottom)
                                  0.25 (on-top-of ?top table)))
```

```
 )
)
(define (problem bw-nc-pc-21)
 (:domain bw-nc-pc-21)
 (:objects block0 block1 block2 block3 block4 block5 block6
      block7 block8 block9 block10 block11 block12 block13 block14
                 block15 block16 block17 block18 block19 block20 - block)
 (:init
  (on-top-of block0 block1)
  (on-top-of block1 block2)
  (on-top-of block2 block3)
  (on-top-of block3 block4)
  (on-top-of block4 table)
  (on-top-of block5 block6)
  (on-top-of block6 block7)
  (on-top-of block7 block8)
  (on-top-of block8 table)
  (on-top-of block9 block10)
  (on-top-of block10 block11)
  (on-top-of block11 block12)
  (on-top-of block12 block13)
  (on-top-of block13 block14)
  (on-top-of block14 block15)
  (on-top-of block15 block16)
  (on-top-of block16 block17)
  (on-top-of block17 table)
  (on-top-of block18 table)
  (on-top-of block19 table)
  (on-top-of block20 table)
 )
 (:goal
  (and
   (on-top-of block0 block5)
   (on-top-of block5 block3)
   (on-top-of block3 block17)
   (on-top-of block17 block19)
   (on-top-of block19 block9)
   (on-top-of block9 block12)
   (on-top-of block12 block1)
   (on-top-of block1 table)
   (on-top-of block10 block8)
   (on-top-of block8 block15)
   (on-top-of block15 block7)
   (on-top-of block7 block20)
   (on-top-of block20 block13)
   (on-top-of block13 block2)
   (on-top-of block2 block6)
   (on-top-of block6 block4)
   (on-top-of block4 block14)
```

```
   (on-top-of block14 block18)
   (on-top-of block18 block11)
   (on-top-of block11 table)
   (on-top-of block16 table)

  )
 )
 (:goal-reward 500)
)
```

D.7 bw-nc-pc-5

```
(define (domain bw-nc-pc-5)
 (:requirements :adl :probabilistic-effects :fluents :rewards)
 (:types block table)
 (:constants table - table)
 (:predicates
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
                     (on-top-of ?top ?bottom)
                     (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (decrease (reward) 1)
              (probabilistic 0.75 (and (holding ?top)
                                       (not (on-top-of ?top ?bottom)))
                             0.25 (when (not (= ?bottom table))
                                        (and (not (on-top-of ?top ?bottom))
                                             (on-top-of ?top table)))))
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
  :effect                 (and (not (holding ?top))
              (probabilistic 0.75 (on-top-of ?top ?bottom)
                             0.25 (on-top-of ?top table)))
 )
)
(define (problem bw-nc-pc-5)
 (:domain bw-nc-pc-5)
 (:objects block0 block1 block2 block3 block4 - block)
 (:init
```

```
  (on-top-of block0 table)
  (on-top-of block1 table)
  (on-top-of block2 block3)
  (on-top-of block3 table)
  (on-top-of block4 table)
 )
 (:goal
  (and
   (on-top-of block0 table)
   (on-top-of block3 block2)
   (on-top-of block2 block1)
   (on-top-of block1 block4)
   (on-top-of block4 table)

  )
 )
 (:goal-reward 500)
)
```

D.8 bw-nc-pc-8

```
(define (domain bw-nc-pc-8)
 (:requirements :adl :probabilistic-effects :fluents :rewards)
 (:types block table)
 (:constants table - table)
 (:predicates
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
                     (on-top-of ?top ?bottom)
                     (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (decrease (reward) 1)
               (probabilistic 0.75 (and (holding ?top)
                                        (not (on-top-of ?top ?bottom)))
                                   0.25 (when (not (= ?bottom table))
                                              (and (not (on-top-of ?top ?bottom))
                                                   (on-top-of ?top table)))))
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
```

```
  :effect                 (and (not (holding ?top))
              (probabilistic 0.75 (on-top-of ?top ?bottom)
                             0.25 (on-top-of ?top table)))
 )
)
(define (problem bw-nc-pc-8)
 (:domain bw-nc-pc-8)
 (:objects block0 block1 block2 block3 block4 block5 block6 block7 - block)
 (:init
  (on-top-of block0 block1)
  (on-top-of block1 table)
  (on-top-of block2 table)
  (on-top-of block3 block4)
  (on-top-of block4 block5)
  (on-top-of block5 table)
  (on-top-of block6 table)
  (on-top-of block7 table)
 )
 (:goal
  (and
   (on-top-of block0 block6)
   (on-top-of block6 block4)
   (on-top-of block4 block2)
   (on-top-of block2 block1)
   (on-top-of block1 table)
   (on-top-of block3 block7)
   (on-top-of block7 table)
   (on-top-of block5 table)

  )
 )
 (:goal-reward 500)
)
```

D.9 bw-nc-pc-nr-8

```
(define (domain bw-nc-pc-nr-8)
 (:requirements :adl :probabilistic-effects :fluents)
 (:types block table)
 (:constants table - table)
 (:predicates
  (holding ?block - block)
  (on-top-of ?block - block ?obj)
 )
 (:action pick-up-block-from
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (forall (?b - block) (not (holding ?b)))
```

```
                    (on-top-of ?top ?bottom)
                    (forall (?b - block) (not (on-top-of ?b ?top))))
  :effect                (and (probabilistic 0.75 (and (holding ?top)
                                                       (not (on-top-of ?top ?bottom)))
                                              0.25 (when (not (= ?bottom table))
                                                         (and (not (on-top-of ?top ?bottom))
                                                              (on-top-of ?top table)))))
 )
 (:action put-down-block-on
  :parameters (?top - block ?bottom)
  :precondition (and (not (= ?top ?bottom))
                     (holding ?top)
                     (or (= ?bottom table)
                         (forall (?b - block) (not (on-top-of ?b ?bottom)))))
  :effect                 (and (not (holding ?top))
               (probabilistic 0.75 (on-top-of ?top ?bottom)
                              0.25 (on-top-of ?top table)))
 )
)
(define (problem bw-nc-pc-nr-8)
 (:domain bw-nc-pc-nr-8)
 (:objects block0 block1 block2 block3 block4 block5 block6 block7 - block)
 (:init
  (on-top-of block0 block1)
  (on-top-of block1 table)
  (on-top-of block2 table)
  (on-top-of block3 block4)
  (on-top-of block4 block5)
  (on-top-of block5 table)
  (on-top-of block6 table)
  (on-top-of block7 table)
 )
 (:goal
  (and
   (on-top-of block0 block6)
   (on-top-of block6 block4)
   (on-top-of block4 block2)
   (on-top-of block2 block1)
   (on-top-of block1 table)
   (on-top-of block3 block7)
   (on-top-of block7 table)
   (on-top-of block5 table)

  )
 )
)
```

D.10 bx-c10-b10-pc-nr

```
(define (domain bx-c10-b10-pc-nr)
 (:requirements :typing :equality :disjunctive-preconditions
                :probabilistic-effects :existential-preconditions
                :conditional-effects :negative-preconditions
                :universal-preconditions)
 (:types city box truck plane)
 (:predicates (box-at-city ?b - box ?c - city)
              (truck-at-city ?t - truck ?c - city)
              (box-on-truck ?b - box ?t - truck)
              (plane-at-city ?p - plane ?c - city)
              (box-on-plane ?b - box ?p - plane)
              (destination ?b - box ?dst - city)
              (can-drive ?src - city ?dst - city)
              (wrong-drive1 ?src - city ?wrongdst - city)
              (wrong-drive2 ?src - city ?wrongdst - city)
              (wrong-drive3 ?src - city ?wrongdst - city)
              (can-fly ?src - city ?dst - city))
 (:action load-box-on-truck-in-city
  :parameters (?b - box ?t - truck ?c - city)
  :precondition (and (box-at-city ?b ?c)
                     (not (destination ?b ?c))
                     (truck-at-city ?t ?c)
                )
  :effect (and (box-on-truck ?b ?t)
               (not (box-at-city ?b ?c))
          )
 )
 (:action unload-box-from-truck-in-city
  :parameters (?b - box ?t - truck ?c - city)
  :precondition (and (box-on-truck ?b ?t)
                     (truck-at-city ?t ?c)
                )
  :effect (and (box-at-city ?b ?c)
               (not (box-on-truck ?b ?t))
          )
 )
 (:action load-box-on-plane-in-city
  :parameters (?b - box ?p - plane ?c - city)
  :precondition (and (box-at-city ?b ?c)
                     (not (destination ?b ?c))
                     (plane-at-city ?p ?c)
                )
  :effect (and (box-on-plane ?b ?p)
               (not (box-at-city ?b ?c))
          )
 )
 (:action unload-box-from-plane-in-city
```

```
  :parameters (?b - box ?p - plane ?c - city)
  :precondition (and (box-on-plane ?b ?p)
                     (plane-at-city ?p ?c)
                )
  :effect (and (box-at-city ?b ?c)
               (not (box-on-plane ?b ?p))
          )
 )
 (:action drive-truck
  :parameters (?t - truck ?src - city ?dst - city)
  :precondition (and (truck-at-city ?t ?src)
                     (can-drive ?src ?dst)
                )
  :effect (and (not (truck-at-city ?t ?src))
               (probabilistic
                0.2 (forall (?wrongdst1 - city)
                    (when (wrong-drive1 ?src ?wrongdst1)
                    (forall (?wrongdst2 - city)
                    (when (wrong-drive2 ?src ?wrongdst2)
                    (forall (?wrongdst3 - city)
                    (when (wrong-drive3 ?src ?wrongdst3)
                     (probabilistic
                      1/3 (truck-at-city ?t ?wrongdst1)
                      1/3 (truck-at-city ?t ?wrongdst2)
                      1/3 (truck-at-city ?t ?wrongdst3)
                     )
                    ))))))
                0.8 (truck-at-city ?t ?dst)
               )
          )
 )
 (:action fly-plane
  :parameters (?p - plane ?src - city ?dst - city)
  :precondition (and (plane-at-city ?p ?src)
                     (can-fly ?src ?dst)
                )
  :effect (and (not (plane-at-city ?p ?src))
               (plane-at-city ?p ?dst)
          )
 )
)
(define
 (problem bx-c10-b10-pc-nr)
  (:domain bx-c10-b10-pc-nr)
  (:objects box0 - box
            box1 - box
            box2 - box
            box3 - box
            box4 - box
```

```
            box5 - box
            box6 - box
            box7 - box
            box8 - box
            box9 - box
            truck0 - truck
            truck1 - truck
            plane0 - plane
            truck2 - truck
            truck3 - truck
            plane1 - plane
            city0 - city
            city1 - city
            city2 - city
            city3 - city
            city4 - city
            city5 - city
            city6 - city
            city7 - city
            city8 - city
            city9 - city
)
(:init (box-at-city box0 city2)
       (destination box0 city4)
       (box-at-city box1 city9)
       (destination box1 city3)
       (box-at-city box2 city8)
       (destination box2 city1)
       (box-at-city box3 city0)
       (destination box3 city3)
       (box-at-city box4 city1)
       (destination box4 city0)
       (box-at-city box5 city1)
       (destination box5 city8)
       (box-at-city box6 city7)
       (destination box6 city6)
       (box-at-city box7 city1)
       (destination box7 city5)
       (box-at-city box8 city5)
       (destination box8 city7)
       (box-at-city box9 city8)
       (destination box9 city9)
       (truck-at-city truck0 city0)
       (truck-at-city truck1 city0)
       (plane-at-city plane0 city0)
       (truck-at-city truck2 city1)
       (truck-at-city truck3 city1)
       (plane-at-city plane1 city1)
       (can-drive city0 city9)
```

```
(can-drive city0 city8)
(can-drive city0 city6)
(wrong-drive1 city0 city9)
(wrong-drive2 city0 city8)
(wrong-drive3 city0 city6)
(can-fly city0 city1)
(can-drive city1 city7)
(can-drive city1 city5)
(can-drive city1 city2)
(wrong-drive1 city1 city7)
(wrong-drive2 city1 city5)
(wrong-drive3 city1 city2)
(can-fly city1 city0)
(can-drive city2 city1)
(can-drive city2 city6)
(can-drive city2 city4)
(can-drive city2 city5)
(can-drive city2 city7)
(can-drive city2 city8)
(wrong-drive1 city2 city1)
(wrong-drive2 city2 city6)
(wrong-drive3 city2 city4)
(can-drive city3 city4)
(can-drive city3 city5)
(can-drive city3 city6)
(wrong-drive1 city3 city4)
(wrong-drive2 city3 city5)
(wrong-drive3 city3 city6)
(can-drive city4 city2)
(can-drive city4 city3)
(can-drive city4 city6)
(can-drive city4 city5)
(wrong-drive1 city4 city2)
(wrong-drive2 city4 city3)
(wrong-drive3 city4 city6)
(can-drive city5 city1)
(can-drive city5 city2)
(can-drive city5 city3)
(can-drive city5 city4)
(can-drive city5 city7)
(can-drive city5 city6)
(wrong-drive1 city5 city1)
(wrong-drive2 city5 city2)
(wrong-drive3 city5 city3)
(can-drive city6 city0)
(can-drive city6 city2)
(can-drive city6 city3)
(can-drive city6 city4)
(can-drive city6 city5)
```

```
        (can-drive city6 city9)
        (wrong-drive1 city6 city0)
        (wrong-drive2 city6 city2)
        (wrong-drive3 city6 city3)
        (can-drive city7 city1)
        (can-drive city7 city5)
        (can-drive city7 city2)
        (wrong-drive1 city7 city1)
        (wrong-drive2 city7 city5)
        (wrong-drive3 city7 city2)
        (can-drive city8 city0)
        (can-drive city8 city9)
        (can-drive city8 city2)
        (wrong-drive1 city8 city0)
        (wrong-drive2 city8 city9)
        (wrong-drive3 city8 city2)
        (can-drive city9 city0)
        (can-drive city9 city8)
        (can-drive city9 city6)
        (wrong-drive1 city9 city0)
        (wrong-drive2 city9 city8)
        (wrong-drive3 city9 city6)
  )
  (:goal (forall (?b - box)
                 (exists (?c - city)
                         (and (destination ?b ?c)
                              (box-at-city ?b ?c)
                         )
                 )
         )
  )
)
```

D.11 bx-c10-b10-pc

```
(define (domain bx-c10-b10-pc)
 (:requirements :typing :equality :disjunctive-preconditions
                :probabilistic-effects :existential-preconditions
                :conditional-effects :negative-preconditions
                :universal-preconditions :rewards)
 (:types city box truck plane)
 (:predicates (box-at-city ?b - box ?c - city)
              (truck-at-city ?t - truck ?c - city)
              (box-on-truck ?b - box ?t - truck)
              (plane-at-city ?p - plane ?c - city)
              (box-on-plane ?b - box ?p - plane)
              (destination ?b - box ?dst - city)
              (can-drive ?src - city ?dst - city)
```

```
             (wrong-drive1 ?src - city ?wrongdst - city)
             (wrong-drive2 ?src - city ?wrongdst - city)
             (wrong-drive3 ?src - city ?wrongdst - city)
             (can-fly ?src - city ?dst - city))
 (:action load-box-on-truck-in-city
  :parameters (?b - box ?t - truck ?c - city)
  :precondition (and (box-at-city ?b ?c)
                     (not (destination ?b ?c))
                     (truck-at-city ?t ?c)
                )
  :effect (and (decrease (reward) 1)
               (box-on-truck ?b ?t)
               (not (box-at-city ?b ?c))
          )
 )
 (:action unload-box-from-truck-in-city
  :parameters (?b - box ?t - truck ?c - city)
  :precondition (and (box-on-truck ?b ?t)
                     (truck-at-city ?t ?c)
                )
  :effect (and (box-at-city ?b ?c)
               (not (box-on-truck ?b ?t))
          )
 )
 (:action load-box-on-plane-in-city
  :parameters (?b - box ?p - plane ?c - city)
  :precondition (and (box-at-city ?b ?c)
                     (not (destination ?b ?c))
                     (plane-at-city ?p ?c)
                )
  :effect (and (decrease (reward) 1)
               (box-on-plane ?b ?p)
               (not (box-at-city ?b ?c))
          )
 )
 (:action unload-box-from-plane-in-city
  :parameters (?b - box ?p - plane ?c - city)
  :precondition (and (box-on-plane ?b ?p)
                     (plane-at-city ?p ?c)
                )
  :effect (and (box-at-city ?b ?c)
               (not (box-on-plane ?b ?p))
          )
 )
 (:action drive-truck
  :parameters (?t - truck ?src - city ?dst - city)
  :precondition (and (truck-at-city ?t ?src)
                     (can-drive ?src ?dst)
                )
```

```
  :effect (and (decrease (reward) 5)
               (not (truck-at-city ?t ?src))
               (probabilistic
                0.2 (forall (?wrongdst1 - city)
                    (when (wrong-drive1 ?src ?wrongdst1)
                    (forall (?wrongdst2 - city)
                    (when (wrong-drive2 ?src ?wrongdst2)
                    (forall (?wrongdst3 - city)
                    (when (wrong-drive3 ?src ?wrongdst3)
                     (probabilistic
                      1/3 (truck-at-city ?t ?wrongdst1)
                      1/3 (truck-at-city ?t ?wrongdst2)
                      1/3 (truck-at-city ?t ?wrongdst3)
                     )
                    ))))))
                0.8 (truck-at-city ?t ?dst)
               )
         )
 )
 (:action fly-plane
  :parameters (?p - plane ?src - city ?dst - city)
  :precondition (and (plane-at-city ?p ?src)
                     (can-fly ?src ?dst)
                )
  :effect (and (decrease (reward) 25)
              (not (plane-at-city ?p ?src))
               (plane-at-city ?p ?dst)
         )
 )
)
(define
 (problem bx-c10-b10-pc)
  (:domain bx-c10-b10-pc)
  (:objects box0 - box
            box1 - box
            box2 - box
            box3 - box
            box4 - box
            box5 - box
            box6 - box
            box7 - box
            box8 - box
            box9 - box
            truck0 - truck
            truck1 - truck
            plane0 - plane
            truck2 - truck
            truck3 - truck
            plane1 - plane
```

```
            city0 - city
            city1 - city
            city2 - city
            city3 - city
            city4 - city
            city5 - city
            city6 - city
            city7 - city
            city8 - city
            city9 - city
)
(:init (box-at-city box0 city2)
       (destination box0 city4)
       (box-at-city box1 city9)
       (destination box1 city3)
       (box-at-city box2 city8)
       (destination box2 city1)
       (box-at-city box3 city0)
       (destination box3 city3)
       (box-at-city box4 city1)
       (destination box4 city0)
       (box-at-city box5 city1)
       (destination box5 city8)
       (box-at-city box6 city7)
       (destination box6 city6)
       (box-at-city box7 city1)
       (destination box7 city5)
       (box-at-city box8 city5)
       (destination box8 city7)
       (box-at-city box9 city8)
       (destination box9 city9)
       (truck-at-city truck0 city0)
       (truck-at-city truck1 city0)
       (plane-at-city plane0 city0)
       (truck-at-city truck2 city1)
       (truck-at-city truck3 city1)
       (plane-at-city plane1 city1)
       (can-drive city0 city9)
       (can-drive city0 city8)
       (can-drive city0 city6)
       (wrong-drive1 city0 city9)
       (wrong-drive2 city0 city8)
       (wrong-drive3 city0 city6)
       (can-fly city0 city1)
       (can-drive city1 city7)
       (can-drive city1 city5)
       (can-drive city1 city2)
       (wrong-drive1 city1 city7)
       (wrong-drive2 city1 city5)
```

```
(wrong-drive3 city1 city2)
(can-fly city1 city0)
(can-drive city2 city1)
(can-drive city2 city6)
(can-drive city2 city4)
(can-drive city2 city5)
(can-drive city2 city7)
(can-drive city2 city8)
(wrong-drive1 city2 city1)
(wrong-drive2 city2 city6)
(wrong-drive3 city2 city4)
(can-drive city3 city4)
(can-drive city3 city5)
(can-drive city3 city6)
(wrong-drive1 city3 city4)
(wrong-drive2 city3 city5)
(wrong-drive3 city3 city6)
(can-drive city4 city2)
(can-drive city4 city3)
(can-drive city4 city6)
(can-drive city4 city5)
(wrong-drive1 city4 city2)
(wrong-drive2 city4 city3)
(wrong-drive3 city4 city6)
(can-drive city5 city1)
(can-drive city5 city2)
(can-drive city5 city3)
(can-drive city5 city4)
(can-drive city5 city7)
(can-drive city5 city6)
(wrong-drive1 city5 city1)
(wrong-drive2 city5 city2)
(wrong-drive3 city5 city3)
(can-drive city6 city0)
(can-drive city6 city2)
(can-drive city6 city3)
(can-drive city6 city4)
(can-drive city6 city5)
(can-drive city6 city9)
(wrong-drive1 city6 city0)
(wrong-drive2 city6 city2)
(wrong-drive3 city6 city3)
(can-drive city7 city1)
(can-drive city7 city5)
(can-drive city7 city2)
(wrong-drive1 city7 city1)
(wrong-drive2 city7 city5)
(wrong-drive3 city7 city2)
(can-drive city8 city0)
```

```
        (can-drive city8 city9)
        (can-drive city8 city2)
        (wrong-drive1 city8 city0)
        (wrong-drive2 city8 city9)
        (wrong-drive3 city8 city2)
        (can-drive city9 city0)
        (can-drive city9 city8)
        (can-drive city9 city6)
        (wrong-drive1 city9 city0)
        (wrong-drive2 city9 city8)
        (wrong-drive3 city9 city6)
 )
 (:goal (forall (?b - box)
                (exists (?c - city)
                        (and (destination ?b ?c)
                             (box-at-city ?b ?c)
                        )
                )
        )
 )
 (:goal-reward 500)
)
```

D.12 bx-c15-b10-pc

```
(define (domain bx-c15-b10-pc)
 (:requirements :typing :equality :disjunctive-preconditions
                :probabilistic-effects :existential-preconditions
                :conditional-effects :negative-preconditions
                :universal-preconditions :rewards)
 (:types city box truck plane)
 (:predicates (box-at-city ?b - box ?c - city)
              (truck-at-city ?t - truck ?c - city)
              (box-on-truck ?b - box ?t - truck)
              (plane-at-city ?p - plane ?c - city)
              (box-on-plane ?b - box ?p - plane)
              (destination ?b - box ?dst - city)
              (can-drive ?src - city ?dst - city)
              (wrong-drive1 ?src - city ?wrongdst - city)
              (wrong-drive2 ?src - city ?wrongdst - city)
              (wrong-drive3 ?src - city ?wrongdst - city)
              (can-fly ?src - city ?dst - city))
 (:action load-box-on-truck-in-city
  :parameters (?b - box ?t - truck ?c - city)
  :precondition (and (box-at-city ?b ?c)
                     (not (destination ?b ?c))
                     (truck-at-city ?t ?c)
                )
```

```
 :effect (and (decrease (reward) 1)
              (box-on-truck ?b ?t)
              (not (box-at-city ?b ?c))
         )
)
(:action unload-box-from-truck-in-city
 :parameters (?b - box ?t - truck ?c - city)
 :precondition (and (box-on-truck ?b ?t)
                    (truck-at-city ?t ?c)
                )
 :effect (and (box-at-city ?b ?c)
              (not (box-on-truck ?b ?t))
         )
)
(:action load-box-on-plane-in-city
 :parameters (?b - box ?p - plane ?c - city)
 :precondition (and (box-at-city ?b ?c)
                    (not (destination ?b ?c))
                    (plane-at-city ?p ?c)
                )
 :effect (and (decrease (reward) 1)
              (box-on-plane ?b ?p)
              (not (box-at-city ?b ?c))
         )
)
(:action unload-box-from-plane-in-city
 :parameters (?b - box ?p - plane ?c - city)
 :precondition (and (box-on-plane ?b ?p)
                    (plane-at-city ?p ?c)
                )
 :effect (and (box-at-city ?b ?c)
              (not (box-on-plane ?b ?p))
         )
)
(:action drive-truck
 :parameters (?t - truck ?src - city ?dst - city)
 :precondition (and (truck-at-city ?t ?src)
                    (can-drive ?src ?dst)
                )
 :effect (and (decrease (reward) 5)
              (not (truck-at-city ?t ?src))
              (probabilistic
               0.2 (forall (?wrongdst1 - city)
                   (when (wrong-drive1 ?src ?wrongdst1)
                   (forall (?wrongdst2 - city)
                   (when (wrong-drive2 ?src ?wrongdst2)
                   (forall (?wrongdst3 - city)
                   (when (wrong-drive3 ?src ?wrongdst3)
                    (probabilistic
```

```
                    1/3 (truck-at-city ?t ?wrongdst1)
                    1/3 (truck-at-city ?t ?wrongdst2)
                    1/3 (truck-at-city ?t ?wrongdst3)
                   )
                  ))))))
               0.8 (truck-at-city ?t ?dst)
              )
         )
 )
 (:action fly-plane
  :parameters (?p - plane ?src - city ?dst - city)
  :precondition (and (plane-at-city ?p ?src)
                     (can-fly ?src ?dst)
                )
  :effect (and (decrease (reward) 25)
               (not (plane-at-city ?p ?src))
               (plane-at-city ?p ?dst)
         )
 )
)
(define
 (problem bx-c15-b10-pc)
  (:domain bx-c15-b10-pc)
  (:objects box0 - box
            box1 - box
            box2 - box
            box3 - box
            box4 - box
            box5 - box
            box6 - box
            box7 - box
            box8 - box
            box9 - box
            truck0 - truck
            truck1 - truck
            plane0 - plane
            truck2 - truck
            truck3 - truck
            plane1 - plane
            city0 - city
            city1 - city
            city2 - city
            city3 - city
            city4 - city
            city5 - city
            city6 - city
            city7 - city
            city8 - city
            city9 - city
```

```
           city10 - city
           city11 - city
           city12 - city
           city13 - city
           city14 - city
)
(:init (box-at-city box0 city11)
       (destination box0 city6)
       (box-at-city box1 city8)
       (destination box1 city12)
       (box-at-city box2 city6)
       (destination box2 city10)
       (box-at-city box3 city0)
       (destination box3 city12)
       (box-at-city box4 city3)
       (destination box4 city4)
       (box-at-city box5 city7)
       (destination box5 city12)
       (box-at-city box6 city0)
       (destination box6 city9)
       (box-at-city box7 city2)
       (destination box7 city9)
       (box-at-city box8 city0)
       (destination box8 city1)
       (box-at-city box9 city0)
       (destination box9 city6)
       (truck-at-city truck0 city0)
       (truck-at-city truck1 city0)
       (plane-at-city plane0 city0)
       (truck-at-city truck2 city1)
       (truck-at-city truck3 city1)
       (plane-at-city plane1 city1)
       (can-drive city0 city14)
       (can-drive city0 city9)
       (can-drive city0 city13)
       (wrong-drive1 city0 city14)
       (wrong-drive2 city0 city9)
       (wrong-drive3 city0 city13)
       (can-fly city0 city1)
       (can-drive city1 city11)
       (can-drive city1 city5)
       (can-drive city1 city10)
       (wrong-drive1 city1 city11)
       (wrong-drive2 city1 city5)
       (wrong-drive3 city1 city10)
       (can-fly city1 city0)
       (can-drive city2 city8)
       (can-drive city2 city12)
       (can-drive city2 city3)
```

```
(wrong-drive1 city2 city8)
(wrong-drive2 city2 city12)
(wrong-drive3 city2 city3)
(can-drive city3 city2)
(can-drive city3 city12)
(can-drive city3 city8)
(can-drive city3 city5)
(wrong-drive1 city3 city2)
(wrong-drive2 city3 city12)
(wrong-drive3 city3 city8)
(can-drive city4 city6)
(can-drive city4 city7)
(can-drive city4 city10)
(wrong-drive1 city4 city6)
(wrong-drive2 city4 city7)
(wrong-drive3 city4 city10)
(can-drive city5 city1)
(can-drive city5 city3)
(can-drive city5 city10)
(wrong-drive1 city5 city1)
(wrong-drive2 city5 city3)
(wrong-drive3 city5 city10)
(can-drive city6 city4)
(can-drive city6 city10)
(can-drive city6 city7)
(can-drive city6 city11)
(wrong-drive1 city6 city4)
(wrong-drive2 city6 city10)
(wrong-drive3 city6 city7)
(can-drive city7 city4)
(can-drive city7 city6)
(can-drive city7 city12)
(can-drive city7 city10)
(wrong-drive1 city7 city4)
(wrong-drive2 city7 city6)
(wrong-drive3 city7 city12)
(can-drive city8 city2)
(can-drive city8 city3)
(can-drive city8 city12)
(wrong-drive1 city8 city2)
(wrong-drive2 city8 city3)
(wrong-drive3 city8 city12)
(can-drive city9 city0)
(can-drive city9 city14)
(can-drive city9 city13)
(wrong-drive1 city9 city0)
(wrong-drive2 city9 city14)
(wrong-drive3 city9 city13)
(can-drive city10 city1)
```

```
        (can-drive city10 city4)
        (can-drive city10 city5)
        (can-drive city10 city6)
        (can-drive city10 city7)
        (can-drive city10 city11)
        (wrong-drive1 city10 city1)
        (wrong-drive2 city10 city4)
        (wrong-drive3 city10 city5)
        (can-drive city11 city1)
        (can-drive city11 city10)
        (can-drive city11 city6)
        (wrong-drive1 city11 city1)
        (wrong-drive2 city11 city10)
        (wrong-drive3 city11 city6)
        (can-drive city12 city2)
        (can-drive city12 city3)
        (can-drive city12 city7)
        (can-drive city12 city8)
        (wrong-drive1 city12 city2)
        (wrong-drive2 city12 city3)
        (wrong-drive3 city12 city7)
        (can-drive city13 city0)
        (can-drive city13 city9)
        (can-drive city13 city14)
        (wrong-drive1 city13 city0)
        (wrong-drive2 city13 city9)
        (wrong-drive3 city13 city14)
        (can-drive city14 city0)
        (can-drive city14 city9)
        (can-drive city14 city13)
        (wrong-drive1 city14 city0)
        (wrong-drive2 city14 city9)
        (wrong-drive3 city14 city13)
  )
  (:goal (forall (?b - box)
                 (exists (?c - city)
                         (and (destination ?b ?c)
                              (box-at-city ?b ?c)
                         )
                 )
         )
  )
  (:goal-reward 500)
)
```

D.13 bx-c5-b10-pc-nr

```
(define (domain bx-c5-b10-pc-nr)
```

```
(:requirements :typing :equality :disjunctive-preconditions
               :probabilistic-effects :existential-preconditions
               :conditional-effects :negative-preconditions
               :universal-preconditions)
(:types city box truck plane)
(:predicates (box-at-city ?b - box ?c - city)
             (truck-at-city ?t - truck ?c - city)
             (box-on-truck ?b - box ?t - truck)
             (plane-at-city ?p - plane ?c - city)
             (box-on-plane ?b - box ?p - plane)
             (destination ?b - box ?dst - city)
             (can-drive ?src - city ?dst - city)
             (wrong-drive1 ?src - city ?wrongdst - city)
             (wrong-drive2 ?src - city ?wrongdst - city)
             (wrong-drive3 ?src - city ?wrongdst - city)
             (can-fly ?src - city ?dst - city))
(:action load-box-on-truck-in-city
 :parameters (?b - box ?t - truck ?c - city)
 :precondition (and (box-at-city ?b ?c)
                    (not (destination ?b ?c))
                    (truck-at-city ?t ?c)
                )
 :effect (and (box-on-truck ?b ?t)
              (not (box-at-city ?b ?c))
         )
)
(:action unload-box-from-truck-in-city
 :parameters (?b - box ?t - truck ?c - city)
 :precondition (and (box-on-truck ?b ?t)
                    (truck-at-city ?t ?c)
                )
 :effect (and (box-at-city ?b ?c)
              (not (box-on-truck ?b ?t))
         )
)
(:action load-box-on-plane-in-city
 :parameters (?b - box ?p - plane ?c - city)
 :precondition (and (box-at-city ?b ?c)
                    (not (destination ?b ?c))
                    (plane-at-city ?p ?c)
                )
 :effect (and (box-on-plane ?b ?p)
              (not (box-at-city ?b ?c))
         )
)
(:action unload-box-from-plane-in-city
 :parameters (?b - box ?p - plane ?c - city)
 :precondition (and (box-on-plane ?b ?p)
                    (plane-at-city ?p ?c)
```

```
                )
  :effect (and (box-at-city ?b ?c)
               (not (box-on-plane ?b ?p))
          )
 )
 (:action drive-truck
  :parameters (?t - truck ?src - city ?dst - city)
  :precondition (and (truck-at-city ?t ?src)
                     (can-drive ?src ?dst)
                )
  :effect (and (not (truck-at-city ?t ?src))
               (probabilistic
                0.2 (forall (?wrongdst1 - city)
                    (when (wrong-drive1 ?src ?wrongdst1)
                    (forall (?wrongdst2 - city)
                    (when (wrong-drive2 ?src ?wrongdst2)
                    (forall (?wrongdst3 - city)
                    (when (wrong-drive3 ?src ?wrongdst3)
                     (probabilistic
                      1/3 (truck-at-city ?t ?wrongdst1)
                      1/3 (truck-at-city ?t ?wrongdst2)
                      1/3 (truck-at-city ?t ?wrongdst3)
                     )
                    ))))))
                0.8 (truck-at-city ?t ?dst)
               )
          )
 )
 (:action fly-plane
  :parameters (?p - plane ?src - city ?dst - city)
  :precondition (and (plane-at-city ?p ?src)
                     (can-fly ?src ?dst)
                )
  :effect (and (not (plane-at-city ?p ?src))
               (plane-at-city ?p ?dst)
          )
 )
)
(define
 (problem bx-c5-b10-pc-nr)
  (:domain bx-c5-b10-pc-nr)
  (:objects box0 - box
            box1 - box
            box2 - box
            box3 - box
            box4 - box
            box5 - box
            box6 - box
            box7 - box
```

```
            box8 - box
            box9 - box
            truck0 - truck
            truck1 - truck
            plane0 - plane
            truck2 - truck
            truck3 - truck
            plane1 - plane
            city0 - city
            city1 - city
            city2 - city
            city3 - city
            city4 - city
)
(:init (box-at-city box0 city0)
       (destination box0 city4)
       (box-at-city box1 city0)
       (destination box1 city4)
       (box-at-city box2 city0)
       (destination box2 city2)
       (box-at-city box3 city1)
       (destination box3 city4)
       (box-at-city box4 city1)
       (destination box4 city3)
       (box-at-city box5 city2)
       (destination box5 city4)
       (box-at-city box6 city4)
       (destination box6 city3)
       (box-at-city box7 city3)
       (destination box7 city1)
       (box-at-city box8 city0)
       (destination box8 city3)
       (box-at-city box9 city1)
       (destination box9 city0)
       (truck-at-city truck0 city0)
       (truck-at-city truck1 city0)
       (plane-at-city plane0 city0)
       (truck-at-city truck2 city1)
       (truck-at-city truck3 city1)
       (plane-at-city plane1 city1)
       (can-drive city0 city3)
       (can-drive city0 city4)
       (can-drive city0 city2)
       (can-drive city0 city1)
       (wrong-drive1 city0 city3)
       (wrong-drive2 city0 city4)
       (wrong-drive3 city0 city2)
       (can-fly city0 city1)
       (can-drive city1 city3)
```

```
        (can-drive city1 city0)
        (can-drive city1 city2)
        (wrong-drive1 city1 city3)
        (wrong-drive2 city1 city0)
        (wrong-drive3 city1 city2)
        (can-fly city1 city0)
        (can-drive city2 city0)
        (can-drive city2 city1)
        (can-drive city2 city4)
        (can-drive city2 city3)
        (wrong-drive1 city2 city0)
        (wrong-drive2 city2 city1)
        (wrong-drive3 city2 city4)
        (can-drive city3 city0)
        (can-drive city3 city1)
        (can-drive city3 city2)
        (can-drive city3 city4)
        (wrong-drive1 city3 city0)
        (wrong-drive2 city3 city1)
        (wrong-drive3 city3 city2)
        (can-drive city4 city0)
        (can-drive city4 city2)
        (can-drive city4 city3)
        (wrong-drive1 city4 city0)
        (wrong-drive2 city4 city2)
        (wrong-drive3 city4 city3)
  )
  (:goal (forall (?b - box)
                 (exists (?c - city)
                         (and (destination ?b ?c)
                              (box-at-city ?b ?c)
                         )
                 )
         )
  )
)
```

D.14 bx-c5-b10-pc

```
(define (domain bx-c5-b10-pc)
 (:requirements :typing :equality :disjunctive-preconditions
                :probabilistic-effects :existential-preconditions
                :conditional-effects :negative-preconditions
                :universal-preconditions :rewards)
 (:types city box truck plane)
 (:predicates (box-at-city ?b - box ?c - city)
              (truck-at-city ?t - truck ?c - city)
              (box-on-truck ?b - box ?t - truck)
```

```
             (plane-at-city ?p - plane ?c - city)
             (box-on-plane ?b - box ?p - plane)
             (destination ?b - box ?dst - city)
             (can-drive ?src - city ?dst - city)
             (wrong-drive1 ?src - city ?wrongdst - city)
             (wrong-drive2 ?src - city ?wrongdst - city)
             (wrong-drive3 ?src - city ?wrongdst - city)
             (can-fly ?src - city ?dst - city))
(:action load-box-on-truck-in-city
 :parameters (?b - box ?t - truck ?c - city)
 :precondition (and (box-at-city ?b ?c)
                    (not (destination ?b ?c))
                    (truck-at-city ?t ?c)
                )
 :effect (and (decrease (reward) 1)
              (box-on-truck ?b ?t)
              (not (box-at-city ?b ?c))
         )
)
(:action unload-box-from-truck-in-city
 :parameters (?b - box ?t - truck ?c - city)
 :precondition (and (box-on-truck ?b ?t)
                    (truck-at-city ?t ?c)
                )
 :effect (and (box-at-city ?b ?c)
              (not (box-on-truck ?b ?t))
         )
)
(:action load-box-on-plane-in-city
 :parameters (?b - box ?p - plane ?c - city)
 :precondition (and (box-at-city ?b ?c)
                    (not (destination ?b ?c))
                    (plane-at-city ?p ?c)
                )
 :effect (and (decrease (reward) 1)
              (box-on-plane ?b ?p)
              (not (box-at-city ?b ?c))
         )
)
(:action unload-box-from-plane-in-city
 :parameters (?b - box ?p - plane ?c - city)
 :precondition (and (box-on-plane ?b ?p)
                    (plane-at-city ?p ?c)
                )
 :effect (and (box-at-city ?b ?c)
              (not (box-on-plane ?b ?p))
         )
)
(:action drive-truck
```

```
  :parameters (?t - truck ?src - city ?dst - city)
  :precondition (and (truck-at-city ?t ?src)
                     (can-drive ?src ?dst)
                )
  :effect (and (decrease (reward) 5)
               (not (truck-at-city ?t ?src))
               (probabilistic
                0.2 (forall (?wrongdst1 - city)
                    (when (wrong-drive1 ?src ?wrongdst1)
                    (forall (?wrongdst2 - city)
                    (when (wrong-drive2 ?src ?wrongdst2)
                    (forall (?wrongdst3 - city)
                    (when (wrong-drive3 ?src ?wrongdst3)
                     (probabilistic
                      1/3 (truck-at-city ?t ?wrongdst1)
                      1/3 (truck-at-city ?t ?wrongdst2)
                      1/3 (truck-at-city ?t ?wrongdst3)
                     )
                    ))))))
                0.8 (truck-at-city ?t ?dst)
               )
         )
 )
 (:action fly-plane
  :parameters (?p - plane ?src - city ?dst - city)
  :precondition (and (plane-at-city ?p ?src)
                     (can-fly ?src ?dst)
                )
  :effect (and (decrease (reward) 25)
              (not (plane-at-city ?p ?src))
               (plane-at-city ?p ?dst)
         )
 )
)
(define
 (problem bx-c5-b10-pc)
  (:domain bx-c5-b10-pc)
  (:objects box0 - box
            box1 - box
            box2 - box
            box3 - box
            box4 - box
            box5 - box
            box6 - box
            box7 - box
            box8 - box
            box9 - box
            truck0 - truck
            truck1 - truck
```

```
           plane0 - plane
           truck2 - truck
           truck3 - truck
           plane1 - plane
           city0 - city
           city1 - city
           city2 - city
           city3 - city
           city4 - city
)
(:init (box-at-city box0 city0)
       (destination box0 city4)
       (box-at-city box1 city0)
       (destination box1 city4)
       (box-at-city box2 city0)
       (destination box2 city2)
       (box-at-city box3 city1)
       (destination box3 city4)
       (box-at-city box4 city1)
       (destination box4 city3)
       (box-at-city box5 city2)
       (destination box5 city4)
       (box-at-city box6 city4)
       (destination box6 city3)
       (box-at-city box7 city3)
       (destination box7 city1)
       (box-at-city box8 city0)
       (destination box8 city3)
       (box-at-city box9 city1)
       (destination box9 city0)
       (truck-at-city truck0 city0)
       (truck-at-city truck1 city0)
       (plane-at-city plane0 city0)
       (truck-at-city truck2 city1)
       (truck-at-city truck3 city1)
       (plane-at-city plane1 city1)
       (can-drive city0 city3)
       (can-drive city0 city4)
       (can-drive city0 city2)
       (can-drive city0 city1)
       (wrong-drive1 city0 city3)
       (wrong-drive2 city0 city4)
       (wrong-drive3 city0 city2)
       (can-fly city0 city1)
       (can-drive city1 city3)
       (can-drive city1 city0)
       (can-drive city1 city2)
       (wrong-drive1 city1 city3)
       (wrong-drive2 city1 city0)
```

```
        (wrong-drive3 city1 city2)
        (can-fly city1 city0)
        (can-drive city2 city0)
        (can-drive city2 city1)
        (can-drive city2 city4)
        (can-drive city2 city3)
        (wrong-drive1 city2 city0)
        (wrong-drive2 city2 city1)
        (wrong-drive3 city2 city4)
        (can-drive city3 city0)
        (can-drive city3 city1)
        (can-drive city3 city2)
        (can-drive city3 city4)
        (wrong-drive1 city3 city0)
        (wrong-drive2 city3 city1)
        (wrong-drive3 city3 city2)
        (can-drive city4 city0)
        (can-drive city4 city2)
        (can-drive city4 city3)
        (wrong-drive1 city4 city0)
        (wrong-drive2 city4 city2)
        (wrong-drive3 city4 city3)
  )
  (:goal (forall (?b - box)
                 (exists (?c - city)
                         (and (destination ?b ?c)
                              (box-at-city ?b ?c)
                         )
                 )
         )
  )
  (:goal-reward 500)
)
```

D.15 explodingbw-pre

```
(define (domain exploding-blocks-world-pre)
(:requirements  :strips :typing :probabilistic-effects
:negative-preconditions :universal-preconditions
:conditional-effects :equality)
(:types block)
(:predicates (detonated ?b - block)
     (destroyed ?b - block)
     (destroyed-table)
     (holding ?b-block)
     (on-top-of ?b1 - block ?b2 - block)
     (on-top-of-table ?b-block)
)
```

```
(:action pick-up-block
  :parameters (?b - block)
  :precondition (and (not (destroyed ?b))
   (not (destroyed-table))
         (forall (?x - block) (not (on-top-of ?x ?b)))
         (forall (?x - block) (not (holding ?x))))
  :effect (and (holding ?b)
          (forall (?x - block) (not (on-top-of ?b ?x))
      ))
)

(:action put-down-block-on-table
  :parameters (?b - block)
  :precondition (and (holding ?b)
   (not (destroyed-table))
    )
  :effect (and (not (holding ?b))
          (on-top-of-table ?b)
   (when (not (detonated ?b))
       (probabilistic .3 (and (detonated ?b)
  (destroyed-table))
   )))
)

(:action put-down-block-on-block
  :parameters (?b1 - block ?b2 - block)
  :precondition (and (holding ?b1) (not (destroyed ?b2))
   (not (destroyed-table))
   (not (= ?b1 ?b2))
(forall (?x - block) (not (on-top-of ?x ?b2)))
)
  :effect (and (not (holding ?b1))
    (when (not (detonated ?b1))
               (probabilistic .3 (and (detonated ?b1)
               (destroyed ?b2)
   (when (on-top-of-table ?b2)
 (and (on-top-of-table ?b1)
      (not (on-top-of-table ?b2))))
   (forall (?x - block)
      (when (on-top-of ?b2 ?x)
(and (not (on-top-of ?b2 ?x))
     (on-top-of ?b1 ?x))))
)
   .7 (on-top-of ?b1 ?b2)
   ))
   (when (detonated ?b1) (on-top-of ?b1 ?b2))
))
)
```

```
(define (problem exploding-blocks-prob-pre)
(:domain exploding-blocks-world-pre)
(:objects b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 - block)
(:init  (on-top-of b0 b1)
(on-top-of b1 b3)
(on-top-of b3 b7)
(on-top-of-table b7)
(on-top-of b2 b4)
(on-top-of-table b4)
(on-top-of-table b5)
(on-top-of b6 b8)
(on-top-of-table b8)
(on-top-of-table b9)
(on-top-of-table b10)
)
(:goal (and  (on-top-of b7 b3)
(on-top-of b0 b1)
(on-top-of-table b1)
(on-top-of b3 b5)
(on-top-of-table b5)
(on-top-of b6 b2)
)
)
)
```

D.16 file-world-pre

```
(define (domain file-world-pre)
(:requirements :disjunctive-preconditions :negative-preconditions
:conditional-effects :rewards
:probabilistic-effects :universal-preconditions)
(:predicates (has-type ?p)
(goes-in ?p ?f)
(filed ?p)
(haveF0)
(haveF1)
(haveF2)
(haveF3)
(haveF4)
)
(:constants F0 F1 F2 F3 F4 )

(:action get-type
:parameters (?p)
:precondition (and (not (has-type ?p)))
:effect
(and (has-type ?p)
(probabilistic
```

```
0.2 (goes-in ?p F0)
0.2 (goes-in ?p F1)
0.2 (goes-in ?p F2)
0.2 (goes-in ?p F3)
0.2 (goes-in ?p F4)
))
)

(:action get-folder-F0
:precondition  (and  (not (haveF0))
(not (haveF1))
(not (haveF2))
(not (haveF3))
(not (haveF4))
)
:effect (and (decrease (reward) 100)
(haveF0)
))

(:action get-folder-F1
:precondition  (and  (not (haveF0))
(not (haveF1))
(not (haveF2))
(not (haveF3))
(not (haveF4))
)
:effect (and (decrease (reward) 100)
(haveF1)
))

(:action get-folder-F2
:precondition  (and  (not (haveF0))
(not (haveF1))
(not (haveF2))
(not (haveF3))
(not (haveF4))
)
:effect (and (decrease (reward) 100)
(haveF2)
))

(:action get-folder-F3
:precondition  (and  (not (haveF0))
(not (haveF1))
(not (haveF2))
(not (haveF3))
(not (haveF4))
)
:effect (and (decrease (reward) 100)
```

```
(haveF3)
))

(:action get-folder-F4
:precondition  (and  (not (haveF0))
(not (haveF1))
(not (haveF2))
(not (haveF3))
(not (haveF4))
)
:effect (and (decrease (reward) 100)
(haveF4)
))

(:action file-F0
:parameters (?p)
:precondition (and (haveF0) (has-type ?p) (goes-in ?p F0))
:effect (and (decrease (reward) 1)
(filed ?p)
))

(:action file-F1
:parameters (?p)
:precondition (and (haveF1) (has-type ?p) (goes-in ?p F1))
:effect (and (decrease (reward) 1)
(filed ?p)
))

(:action file-F2
:parameters (?p)
:precondition (and (haveF2) (has-type ?p) (goes-in ?p F2))
:effect (and (decrease (reward) 1)
(filed ?p)
))

(:action file-F3
:parameters (?p)
:precondition (and (haveF3) (has-type ?p) (goes-in ?p F3))
:effect (and (decrease (reward) 1)
(filed ?p)
))

(:action file-F4
:parameters (?p)
:precondition (and (haveF4) (has-type ?p) (goes-in ?p F4))
:effect (and (decrease (reward) 1)
(filed ?p)
))
```

```
(:action return-F0
:precondition (haveF0)
:effect
(not (haveF0))
)

(:action return-F1
:precondition (haveF1)
:effect
(not (haveF1))
)

(:action return-F2
:precondition (haveF2)
:effect
(not (haveF2))
)

(:action return-F3
:precondition (haveF3)
:effect
(not (haveF3))
)

(:action return-F4
:precondition (haveF4)
:effect
(not (haveF4))
)

)
(define (problem file-prob-pre)
(:domain file-world-pre)
(:objects p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15 p16
                    p17 p18 p19 p20 p21 p22 p23 p24 p25 p26 p27 p28 p29 )
(:goal (and (filed p0) (filed p1) (filed p2) (filed p3) (filed p4) (filed p5)
     (filed p6) (filed p7) (filed p8) (filed p9) (filed p10) (filed p11)
     (filed p12) (filed p13) (filed p14) (filed p15) (filed p16) (filed p17)
     (filed p18) (filed p19) (filed p20) (filed p21) (filed p22) (filed p23)
     (filed p24) (filed p25) (filed p26) (filed p27) (filed p28) (filed p29) ))
(:goal-reward 600)
)
```

D.17 g-tire-world-pre

```
(define (domain g-tire-world-pre)
(:requirements :typing :strips :negative-preconditions
:probabilistic-effects :disjunctive-preconditions
```

```
:conditional-effects)
(:types location)
(:predicates (vehicle-at ?loc - location)
(hasspare-vehicle)
(hasspare-location ?loc - location)
(road ?from - location ?to - location)
(flattire))
(:action mov-car
:parameters (?from - location ?to - location)
:precondition (and (vehicle-at ?from) (road ?from ?to)
(not (flattire)))
:effect (and (vehicle-at ?to) (not (vehicle-at ?from))
(probabilistic .15 (flattire)))
)

(:action loadtire
:parameters (?loc - location)
:precondition (and (vehicle-at ?loc) (hasspare-location ?loc)
(not (hasspare-vehicle)))
:effect (and (hasspare-vehicle) (not (hasspare-location ?loc)))
)

(:action changetire
:precondition (and (hasspare-vehicle) (flattire))
:effect (and (not (hasspare-vehicle))
 (not (flattire))
  )
)

)

(define (problem g-tire-problem-pre)
(:domain g-tire-world-pre)
(:objects c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf cg ch ci
                                                        cj ck cl cm cn
    d0 d1 d2 d3 d4 d5 d6 - location)
(:init
(vehicle-at c0)
(road c0 c1)
(road c1 c2)
(road c2 c3)
(road c3 c4)
(road c4 c5)
(road c5 c6)
(road c6 c7)
(road c7 c8)
(road c8 c9)
(road c0 ca)
(road ca cb)
```

```
(road cb cc)
(road cc cd)
(road cb ce)
(road cd ce)
(road ce cf)
(road cf d5)
(road d5 d6)
(road d6 cg)
(road cf cg)
(road cg ch)
(road ch ci)
(road ci cj)
(road cj ck)
(road ck cl)
(road cl c9)
(road ch cm)
(road cm cn)
(road cn ci)
(road c6 d0)
(road d0 d1)
(road d1 d2)
(road d2 d3)
(road d3 d4)
(road d4 c8)
(hasspare-location cc)
(hasspare-location d5)
(hasspare-location ck)
(hasspare-location cm)
(hasspare-location d4)
)
(:goal (vehicle-at c9))
)
```

D.18 r-tire-world-pre

```
(define (domain r-tire-world-pre)
(:requirements :typing :strips :rewards :negative-preconditions
:probabilistic-effects :disjunctive-preconditions
:conditional-effects)
(:types location)
(:predicates (vehicle-at ?loc - location)
(hasspare-vehicle)
(hasspare-location ?loc - location)
(road ?from - location ?to - location)
(flattire))
(:action mov-car
:parameters (?from - location ?to - location)
:precondition (and (vehicle-at ?from) (road ?from ?to)
```

```
(not (flattire)))
:effect (and (vehicle-at ?to) (not (vehicle-at ?from))
(decrease reward 1)
(probabilistic .15 (flattire)))
)

(:action loadtire
:parameters (?loc - location)
:precondition (and (vehicle-at ?loc) (hasspare-location ?loc)
(not (hasspare-vehicle)))
:effect (and (hasspare-vehicle) (not (hasspare-location ?loc))
(decrease reward 1))
)

(:action changetire
:precondition (and (hasspare-vehicle) (flattire))
:effect (and (decrease (reward) 1)
 (not (hasspare-vehicle))
 (not (flattire))
  )
)

(:action callAAA
:precondition (flattire)
:effect  (and (decrease (reward) 100)
  (not (flattire))
   )
)
)

(define (problem r-tire-problem-pre)
(:domain r-tire-world-pre)
(:objects c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf cg ch ci
                                                        cj ck cl cm cn
    d0 d1 d2 d3 d4 d5 d6 - location)
(:init
(vehicle-at c0)
(road c0 c1)
(road c1 c2)
(road c2 c3)
(road c3 c4)
(road c4 c5)
(road c5 c6)
(road c6 c7)
(road c7 c8)
(road c8 c9)
(road c0 ca)
(road ca cb)
(road cb cc)
```

```
(road cc cd)
(road cb ce)
(road cd ce)
(road ce cf)
(road cf d5)
(road d5 d6)
(road d6 cg)
(road cf cg)
(road cg ch)
(road ch ci)
(road ci cj)
(road cj ck)
(road ck cl)
(road cl c9)
(road ch cm)
(road cm cn)
(road cn ci)
(road c6 d0)
(road d0 d1)
(road d1 d2)
(road d2 d3)
(road d3 d4)
(road d4 c8)
(hasspare-location cc)
(hasspare-location d5)
(hasspare-location ck)
(hasspare-location cm)
(hasspare-location d4)
)
(:goal (vehicle-at c9))
(:goal-reward 100)
)
```

D.19 towers-of-hanoise-pre

```
(define (domain towers-of-hanoise-pre)
(:requirements :strips :negative-preconditions
   :probabilistic-effects :conditional-effects
   :disjunctive-preconditions :typing
   :universal-preconditions :equality
)
(:types disk stick)
(:predicates
(on-top-of ?d1 - disk ?d2 - disk)
(on-stick ?d1 - disk ?s1 - stick)
(bigger ?d1 - disk ?d2 - disk)
(is-big ?d)
(moved-big)
```

```
(failed)
)

(:action single-move-big-not-moved
:parameters (?d - disk ?d2 - disk ?s1 - stick ?s2 - stick)
:precondition (and  (not (failed))
(not (= ?d ?d2))
(not (= ?s1 ?s2))
      (forall (?x - disk) (not (on-top-of ?x ?d)))
(on-stick ?d ?s1)
(or (on-top-of ?d ?d2) (forall (?x - disk)
(not (on-top-of ?d ?x))))
(not (moved-big))
(forall (?x - disk) (not (and (on-stick ?x ?s2)
(bigger ?d ?x))))
  )
:effect  (and
(probabilistic .99 (and
  (when (on-top-of ?d ?d2) (not (on-top-of ?d ?d2)))
  (forall (?x - disk)
    (when (and (on-stick ?x ?s2)
(forall (?y - disk) (not (on-top-of ?y ?x))))
(on-top-of ?d ?x)
    )
  )
  (not (on-stick ?d ?s1)) (on-stick ?d ?s2)
)
   .01 (failed)
)
(when (is-big ?d) (moved-big))
   )
)

(:action single-move-big-moved
:parameters (?d - disk ?d2 - disk ?s1 - stick ?s2 - stick)
:precondition (and  (not (failed))
(not (= ?d ?d2))
(not (= ?s1 ?s2))
(moved-big)
(forall (?x - disk) (not (on-top-of ?x ?d)))
(on-stick ?d ?s1)
(or (on-top-of ?d ?d2) (forall (?x - disk)
(not (on-top-of ?d ?x))))
(forall (?x - disk) (not (and (on-stick ?x ?s2)
(bigger ?d ?x))))
  )
:effect  (probabilistic .95 (and
  (when (on-top-of ?d ?d2) (not (on-top-of ?d ?d2)))
  (forall (?x - disk)
```

```
    (when (and (on-stick ?x ?s2)
(forall (?y - disk) (not (on-top-of ?y ?x))))
(on-top-of ?d ?x)
    )
  )
  (not (on-stick ?d ?s1)) (on-stick ?d ?s2)
)
.05 (failed)
   )
)

(:action double-move-big-not-moved
:parameters (?d1 - disk ?d2 - disk ?d3 - disk ?s1 - stick ?s2 - stick)
:precondition (and (not (failed))
(not (= ?s1 ?s2))
(not (moved-big))
(on-stick ?d1 ?s1)
(on-stick ?d2 ?s1)
(on-top-of ?d1 ?d2)
(forall (?x - disk) (not (on-top-of ?x ?d1)))
(or (on-top-of ?d2 ?d3) (forall (?x - disk)
(not (on-top-of ?d2 ?x))))
(forall (?x - disk) (not (and (on-stick ?x ?s2)
(bigger ?d2 ?x))))
)
:effect (and
(probabilistic .8 (and (when (on-top-of ?d2 ?d3)
(not (on-top-of ?d2 ?d3)))
(forall (?x - disk)
  (when (and (on-stick ?x ?s2)
(forall (?y - disk) (not (on-top-of ?y ?x))))
(on-top-of ?d2 ?x)
  ))
(not (on-stick ?d1 ?s1))
(not (on-stick ?d2 ?s1))
(on-stick ?d1 ?s2)
(on-stick ?d2 ?s2)
   )
.2 (failed)
   )
(when (is-big ?d2) (moved-big))
)
)

(:action double-move-big-moved
:parameters (?d1 - disk ?d2 - disk ?d3 - disk ?s1 - stick ?s2 - stick)
:precondition (and (not (failed))
(not (= ?s1 ?s2))
(moved-big)
```

```
(on-stick ?d1 ?s1)
(on-stick ?d2 ?s1)
(on-top-of ?d1 ?d2)
(forall (?x - disk) (not (on-top-of ?x ?d1)))
(or (on-top-of ?d2 ?d3) (forall (?x - disk)
(not (on-top-of ?d2 ?x))))
(forall (?x - disk) (not (and (on-stick ?x ?s2)
(bigger ?d2 ?x))))
)
:effect (probabilistic .9 (and (when (on-top-of ?d2 ?d3)
(not (on-top-of ?d2 ?d3)))
(forall (?x - disk)
  (when (and (on-stick ?x ?s2)
(forall (?y - disk) (not (on-top-of ?y ?x))))
(on-top-of ?d2 ?x)
  ))
   (not (on-stick ?d1 ?s1))
(not (on-stick ?d2 ?s1))
(on-stick ?d1 ?s2)
(on-stick ?d2 ?s2)
)
.1 (failed)
  )
)
)

(define (problem toh-prob-pre)
(:domain towers-of-hanoise-pre)
(:objects d1 d2 d3 d4 d5 - disk s1 s2 s3 - stick)
(:init
(on-top-of d1 d2)
(on-top-of d2 d3)
(on-top-of d3 d4)
(on-top-of d4 d5)
(on-stick d1 s1)
(on-stick d2 s1)
(on-stick d3 s1)
(on-stick d4 s1)
(on-stick d5 s1)
(bigger d2 d1)
(bigger d3 d2)
(bigger d3 d1)
(bigger d4 d3)
(bigger d4 d2)
(bigger d4 d1)
(bigger d5 d4)
(bigger d5 d3)
(bigger d5 d2)
(bigger d5 d1)
```

```
(is-big d5)
)
(:goal (and (on-stick d5 s3)
(on-top-of d1 d2)
(on-top-of d2 d3)
(on-top-of d3 d4)
(on-top-of d4 d5)
))
)
```

D.20 zeno-pc

```
(define (domain zeno-travel)
  (:requirements :typing :negative-preconditions :universal-preconditions
 :probabilistic-effects)
  (:types aircraft person city flevel - object)
  (:predicates (at ?x - (either person aircraft) ?c - city)
       (boarding ?p - person ?a - aircraft)
       (in ?p - person ?a - aircraft)
       (debarking ?p -person ?a - aircraft)
       (fuel-level ?a - aircraft ?l - flevel)
       (next ?l1 ?l2 - flevel)
       (flying ?a - aircraft ?c - city)
       (zooming ?a - aircraft ?c - city)
       (refueling ?a - aircraft))

  (:action start-boarding
   :parameters (?p - person ?a - aircraft ?c - city)
   :precondition (and (at ?p ?c) (at ?a ?c))
   :effect (and (not (at ?p ?c)) (boarding ?p ?a)))

  (:action complete-boarding
   :parameters (?p - person ?a - aircraft ?c - city)
   :precondition (and (boarding ?p ?a) (at ?a ?c))
   :effect (probabilistic 1/20 (and (not (boarding ?p ?a))
    (in ?p ?a))))

  (:action start-debarking
   :parameters (?p - person ?a - aircraft ?c - city)
   :precondition (and (in ?p ?a) (at ?a ?c))
   :effect (and (not (in ?p ?a)) (debarking ?p ?a)))

  (:action complete-debarking
   :parameters (?p - person ?a - aircraft ?c - city)
   :precondition (and (debarking ?p ?a) (at ?a ?c))
   :effect (probabilistic 1/30 (and (not (debarking ?p ?a))
    (at ?p ?c))))
```

```
  (:action start-flying
   :parameters (?a - aircraft ?c1 ?c2 - city ?l1 ?l2 - flevel)
   :precondition (and (at ?a ?c1) (fuel-level ?a ?l1) (next ?l2 ?l1)
      (not (refueling ?a))
      (forall (?p - person)
      (and (not (boarding ?p ?a))
   (not (debarking ?p ?a)))))
   :effect (and (not (at ?a ?c1)) (flying ?a ?c2)))

  (:action complete-flying
   :parameters (?a - aircraft ?c2 - city ?l1 ?l2 - flevel)
   :precondition (and (flying ?a ?c2) (fuel-level ?a ?l1)
      (next ?l2 ?l1))
   :effect (probabilistic 1/180 (and (not (flying ?a ?c2)) (at ?a ?c2)
     (not (fuel-level ?a ?l1))
     (fuel-level ?a ?l2))))

  (:action start-zooming
   :parameters (?a - aircraft ?c1 ?c2 - city ?l1 ?l2 ?l3 - flevel)
   :precondition (and (at ?a ?c1) (fuel-level ?a ?l1) (next ?l2 ?l1)
      (next ?l3 ?l2) (not (refueling ?a))
      (forall (?p - person)
      (and (not (boarding ?p ?a))
   (not (debarking ?p ?a)))))
   :effect (and (not (at ?a ?c1)) (zooming ?a ?c2)))

  (:action complete-zooming
   :parameters (?a - aircraft ?c2 - city ?l1 ?l2 ?l3 - flevel)
   :precondition (and (zooming ?a ?c2) (fuel-level ?a ?l1)
      (next ?l2 ?l1) (next ?l3 ?l2))
   :effect (probabilistic 1/100 (and (not (zooming ?a ?c2)) (at ?a ?c2)
     (not (fuel-level ?a ?l1))
     (fuel-level ?a ?l3))))

  (:action start-refueling
   :parameters (?a - aircraft ?c - city ?l ?l1 - flevel)
   :precondition (and (at ?a ?c) (not (refueling ?a))
      (fuel-level ?a ?l) (next ?l ?l1))
   :effect (refueling ?a))

  (:action complete-refuling
   :parameters (?a - aircraft ?l ?l1 - flevel)
   :precondition (and (refueling ?a) (fuel-level ?a ?l) (next ?l ?l1))
   :effect (probabilistic 1/73 (and (not (refueling ?a))
    (fuel-level ?a ?l1)
    (not (fuel-level ?a ?l))))))


(define (problem ZTRAVEL-1-2)
```

```
(:domain zeno-travel)
(:objects
plane1 - aircraft
person1 - person
person2 - person
city0 - city
city1 - city
city2 - city
fl0 - flevel
fl1 - flevel
fl2 - flevel
fl3 - flevel
fl4 - flevel
fl5 - flevel
fl6 - flevel
)
(:init
(at plane1 city0)
(fuel-level plane1 fl1)
(at person1 city0)
(at person2 city2)
(next fl0 fl1)
(next fl1 fl2)
(next fl2 fl3)
(next fl3 fl4)
(next fl4 fl5)
(next fl5 fl6)
)
(:goal (and
(at plane1 city1)
(at person1 city0)
(at person2 city2)
))
)
```

Bibliographie

[1] Shapiro (S. C.), *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[2] Minsky (M.), *Semantic Information Processing*. MIT Press, 1968.

[3] Hofstadter (D. R.) et Dennet (D. C.). « The mind's i », 1981.

[4] Fabiani (P.). « Ressac : Recherche et sauvetage par système autonome coopérant au sein d'un réseau d'information et de décision ». http ://www.cert.fr/dcsd/RESSAC/.

[5] Teichteil-Königsbuch (F.) et Fabiani (P.), « Processus décisionnel de markov décomposé et factorisé pour l'optimisation d'une stratégie d'exploration », *Revue d'Intelligence Artificielle*, vol. to appear, 2005.

[6] Régnier (P.), *Algorithmique de la planification en IA*. Editions Cépaduès, 2004.

[7] Nau (D.), Ghallab (M.) et Traverso (P.), *Automated Planning : Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[8] Dijkstra (E. W.), « A note on two problems in connexion with graphs », *Numerische Mathematik*, vol. 1, 1959, p. 269–271.

[9] Hart (P. E.), .Nilsson (N. J.) et Raphael (B.), « A formal basis for the heuristic determination of minimum cost paths », *IEEE Transactions on Systems Science and Cybernetics SSC4*, vol. 2, 1968, p. 100–107.

[10] Blum (A. L.) et Furst (M. L.), « Fast planning through graph analysis », dans *Fourteenth International Joint Conference on Artificial Intelligence*, p. 1636–1642, 1995.

[11] Kautz (H. A.) et Selman (B.), « Planning as satisfiability », dans *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, p. 359–363, 1992.

[12] Chanthery (E.), Barbier (M.) et Farges (J.-L.), « Planning algorithms for autonomous aerial vehicle », dans *16th IFAC World Congress*, Pragues, République Chèque, 2005.

[13] Damiani (S.), Verfaillie (G.) et Charmeau (M.-C.), « A Continuous Anytime Planning Module for an Autonomous Earth Watching Satellite », dans *ICAPS05 Workshop on Planning and Scheduling for Autonomous Systems*, p. 19–28, Monterey, California, USA, 2005.

[14] Puterman (M. L.), *Markov Decision Processes*. John Wiley & Sons, INC, 1994.

[15] Bellman (R.), *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[16] Boutilier (C.), Dean (T.) et Hanks (S.), « Decision-theoretic planning : Structural assumptions and computational leverage », *J.A.I.R.*, vol. 11, 1999, p. 1–94.

[17] Dearden (R.) et Boutilier (C.), « Abstraction and approximate decision-theoretic planning », *Artificial Intelligence*, vol. 89, n° 1–2, 1997, p. 219–283.

[18] Littman (M. L.), Dean (T. L.) et Kaelbling (L. P.), « On the complexity of solving Markov decision problems », dans *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, p. 394–402, Montreal, Québec, Canada, 1995.

[19] Papadimitriou (C.) et Tsisiklis (J. N.), « The complexity of markov decision processes », *Math. Oper. Res.*, vol. 12, n° 3, 1987, p. 441–450.

[20] Verfaillie (G.), Garcia (F.) et Péret (L.), « Deployment and Maintenance of a Constellation of Satellites : a Benchmark », dans *Proceedings of ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information*, Trento, Italy, June 2003.

[21] Teichteil (F.) et Fabiani (P.), « Influence of modeling structure in probabilistic sequential decision problems », *RAIRO Operations Research*, vol. to appear, 2005.

[22] Dean (T.) et Lin (S.-H.), « Decomposition techniques for planning in stochastic domains », dans *Proceedings 14th IJCAI*, p. 1121–1129, San Francisco, CA, 1995.

[23] Hauskrecht (M.), Meuleau (N.), Kaelbling (L. P.) *et al.*, « Hierarchical solution of markov decision processes using macro-actions », dans *Proceedings 14th UAI*, p. 220–229, San Francisco, CA, 1998.

[24] Parr (R.), « Flexible decomposition algorithms for weakly coupled markov decision problems », dans *Proceedings 14th UAI*, p. 422–430, San Francisco, CA, 1998.

[25] Sabbadin (R.), « Graph partitioning techniques for markov decision processes decomposition », dans *Proceedings 15th ECAI*, p. 670–674, Lyon, France, July 2002.

[26] Boutilier (C.), Dearden (R.) et Goldszmidt (M.), « Stochastic dynamic programming with factored representations », *Artificial Intelligence*, vol. 121, n° 1-2, 2000, p. 49–107.

[27] Boutilier (C.), Dearden (R.) et Goldszmidt (M.), « Exploiting structure in policy construction », dans Mellish (C.), éditeur, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, p. 1104–1111, San Francisco, 1995. Morgan Kaufmann.

[28] Boutilier (C.) et Dearden (R.), « Approximating value trees in structured dynamic programming », dans Saitta (L.), éditeur, *Proceedings of the Thirteenth International Conference on Machine Learning*, 1996.

[29] Boutilier (C.), « Correlated action effects in decision theoretic regression », dans *Uncertainty in Artificial Intelligence*, p. 30–37, 1997.

[30] Boutilier (C.), Brafman (R. I.) et Geib (C.), « Structured reachability analysis for Markov decision processes », dans *Uncertainty in Artificial Intelligence*, p. 24–32, 1998.

[31] Hoey (J.), St-Aubin (R.), Hu (A.) et Boutilier (C.), « Optimal and approximate stochastic planning using decision diagrams ». Rapport technique n° TR-2000-05, University of British Columbia, 10 2000.

[32] Hoey (J.), St-Aubin (R.), Hu (A.) et Boutilier (C.), « SPUDD : Stochastic planning using decision diagrams », dans *Fifteenth Conference on Uncertainty in Artificial Intelligence*, p. 279–288, 1999.

[33] St-Aubin (R.), Hoey (J.) et Boutilier (C.), « APRICODD : Approximate policy construction using decision diagrams », dans *NIPS*, p. 1089–1095, 2000.

[34] Zilberstein (S.), Mouaddib (A.) et Arnt (A.). « Dynamic scheduling of progressive processing plans », 2000.

[35] Cassandra (A. R.), *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. Computer science, U. of Illinois, Providence R.I., 1998.

[36] Dutech (A.), « Solving pomdp's using selected past events », dans *Proceedings 14th ECAI*, p. 281–285, Berlin, Germany, July 2000.

[37] Boutilier (C.) et Poole (D.), « Computing optimal policies for partially observable decision processes using compact representations », dans *Proceedings 13th National Conference on Artificial Intelligence*, p. 1168–1175, Portland, Oregon, USA, 1996. AAAI Press / The MIT Press.

[38] Bertsekas (D.) et Tsitsiklis (J.). « Neuro-dynamic programming : an overview », 1995.

[39] Aycard (O.), Charpillet (F.), Fohr (D.) et Mari (J.), « Place learning and recognition using hidden markov models », dans *In proceedings of IEEE/IROS*, p. 1741–1746, 1997.

[40] Chatila (R.) et Laumond (J.), « Position referencing and consistent world modeling for mobile robots », dans *In IEEE International Conference on Robotics and Automation*, p. 138–145, 1985.

[41] Bacchus (F.), Boutilier (C.) et Grove (A.), « Structured solution methods for non-markovian decision processes », dans *Proceedings 14th AAAI*, p. 112–117, Providence, RI, 1997.

[42] Sabbadin (R.), « A possibilistic model for qualitative sequential decision problems under uncertainty in partially observable environments », dans *In Proceedings UAI99*, p. 567–574. Morgan Kaufmann, 1999.

[43] Guestrin (C.), Hauskrecht (M.) et Kveton (B.), « Solving factored mdps with continuous and discrete variables », dans *Proceedings 20th UAI*, Banff, Canada, 2004.

[44] Kveton (B.) et Hauskrecht (M.), « Heuristic refinements of approximate linear programming for factored continuous-state markov decision processes », dans *Proceedings 14th ICAPS*, p. 306–314, Whistler, Canada, 2004.

[45] Chacon (R.) et Ornstein (D.), « A general ergodic theorem », dans *Illinois J. Math. 4*, p. 153–160, 1960.

[46] Fox (B. L.) et Landi (D. M.), *An algorithm for identifying the ergodic subchains and transient states of a stochastic matrix*. Communications of the ACM, 2e édition, 1968.

[47] Duff (I. S.), « A survey of sparse matrix research », dans *Proceedings of the IEEE*, vol. 65, p. 500–535, Prentice Hall, New York, 1977.

[48] Duff, I. S., Erisman, A. M. et Reid, J. K., *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.

[49] Saad (Y.), *Iterative Methods for Sparse Linear Systems*. Society of Industrial and Applied Mathematics, édition second, 2003.

[50] Teichteil (F.). « Stratégie d'exploration pour un aéronef autonome », 2002.

[51] Dietterich (T. G.), « Hierarchical reinforcement learning using the maxq value function decomposition », *Journal of Artificial Intelligence Research*, vol. 13, 2000, p. 227–303.

[52] Hengst (B.), « Discovering hierarchy in reinforcement learning with hexq », dans *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[53] Jonsson (A.) et Barto, « A causal approach to hierarchical decomposition of factored mdps », dans *Proceedings of the Twenty-Second International Conference on Machine Learning ICML 05*, August 2005.

[54] Hengst (B.), « Generating hierarchical structure in reinforcement learning from state variables », dans *PRICAI*, p. 533–543, 2000.

[55] Kim (K.-E.) et Dean (T.), « Solving factored mdps using non-homogeneous partitions », *Artificial Intelligence*, vol. 147, n° 1-2, 2003, p. 225–251.

[56] Laroche (P.), Charpillet (F.) et Schott (R.), « Decomposition of markov decision processes using directed graphs », dans *In Poster session of the European Conference on Planning (ECP'99)*, 1999.

[57] Teichteil-Königsbuch (F.) et Fabiani (P.), « Influence de la structure de modélisation en planification probabiliste », *JEDAI*, 2004.

[58] Spielman (D.) et Teng (S.-H.), « Smoothed analysis : Why the simplex algorithm usually takes polynomial time », *Journal of the ACM*, vol. 51, 2004, p. 385–463.

[59] Lovejoy (W. S.), « A survey of algorithmic methods for partially observed markov decision processes ». Rapport technique n° 28, Annals of Operation Research, 1991.

[60] Cassandra (A. R.), Kaelbling (L. P.) et Littman (M. L.), « Acting optimally in partially observable stochastic domains », dans *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, vol. 2, p. 1023–1028, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[61] Makhorin (A.). « The gnu linear programming kit ». http ://www.gnu.org/software/glpk/glpk.html.

[62] Dean (T.) et Kanazawa (K.), « A model for reasoning about persistence and causation », *Computational Intelligence*, vol. 5(3), 1989, p. 142–150.

[63] Givan (R.) et Dean (T.), « Model minimization, regression, and propositional strips planning », dans *IJCAI*, p. 1163–1168, 1997.

[64] Dean (T.), Givan (R.) et Leach (S. M.), « Model reduction techniques for computing approximately optimal solutions for markov decision processes », dans *UAI '97 : Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, p. 124–131, 1997.

[65] Gérard (P.) et Sigaud (O.), « Yacs : Combining dynamic programming with generalization in classifier systems », dans *Proceedings of LNAI 1996 : Advances in Classifier Systems*, p. 53–69. Springer-Verlag, 2001.

[66] Gérard (P.), Meyer (J.-A.) et sigaud (O.), « Combining latent learning and dynamic programming in macs », *European Journal of Operational Research*, 2005.

[67] Fikes (R.) et Nilsson (N.), « Strips : A new approach to the application of theorem proving to problem solving », *Artificial Intelligence*, vol. 2, 1971, p. 189–208.

[68] Bryant (R. E.), « Graph-based algorithms for boolean function manipulation », *IEEE Transactions on Computers*, vol. 35, n° 8, 1986, p. 677–691.

[69] R.I. Bahar, E.A. Frohm, C.M. Gaona *et al.*, « Algebraic Decision Diagrams and Their Applications », dans *IEEE /ACM International Conference on CAD*, p. 188–191, Santa Clara, California, 1993. IEEE Computer Society Press.

[70] Somenzi (F.). « Cudd : Cu decision diagram package release », 1998.

[71] Bellman (R.), *Adaptive control processes : A guided tour*. Princeton University Press, Princeton, NJ, 1961.

[72] Fox (M.), Ghallab (M.), Infantes (G.) et Long (D.), « Robot introspection through learned hidden markov models », *Artificial Intelligence*, vol. 169, 2005.

[73] Hendrickson (B.) et Leland (R.). « An improved spectral graph partitioning algorithm for mapping parallel computations », 1995.

[74] Bollig (B.), Sauerhoff (M.), Sieling (D.) et Wegener (I.). « Binary decision diagrams ».

[75] Wegener (I.). « BDDs — design, analysis, complexity, and applications ».

[76] Orey (S.), « An ergodic theorem for markov chains », dans *Z. Warscheinlichtkeitstheorie Verw. Gebiete 1*, p. 174–176, 1962.

[77] Orey (S.), *Limit Theorems for Markov Chains Transition Probabilities*. Van Nostrand, New York, 1971.

[78] Davenport (J. H.), Siret (Y.) et Tournier (E.), *Computer algebra : systems and algorithms for algebraic computation*. Academic Press Ltd., London, UK, 1988.

[79] Geddes (K. O.), Czapor (S. R.) et Labahn (G.), *Algorithms for computer algebra*. Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[80] Bauer (C.), Frink (A.) et Kreckel (R.), « Introduction to the ginac framework for symbolic computation within the c++ programming language », *J. Symb. Comput.*, vol. 33, n° 1, 2002, p. 1–12.

[81] Cumming (M.). « Bakery is a c++ framework for creating gnome applications using gtkmm. ». http ://bakery.sourceforge.net/.

[82] Lih-Shyang (K.-Y. C.). « Document-view-presentation pattern », 1999.

[83] W3C. « Extensible markup language (xml) is a simple, very flexible text format derived from sgml (iso 8879). ». http ://www.w3.org/XML/.

[84] Cumming (M.). « Gtkmm is the official c++ interface for the popular gui library gtk+. ». http ://www.gtkmm.org/.

[85] SGI. « Opengl – the industry's foundation for high performance graphics ». http ://www.opengl.org.

[86] Yasofuku (N.). « Gtkglextmm is a c++ wrapper for gtkglext ». http ://gtkglext.sourceforge.net/.

[87] Mampey (R.), « Algorithmes géométriques pour la gestion des terrains ». Rapport technique, ONERA, 2005.

[88] Hoey (J.), St-Aubin (R.), Hu (A.) et Boutilier (C.), « Spudd : Stochastic planning using decision diagrams », dans *Proceedings 15th UAI*, p. 279–288, San Francisco, CA, 1999.

[89] Koller (D.) et Parr (R.), « Computing factored value functions for policies in structured MDPs », dans *IJCAI*, p. 1332–1339. Morgan Kaufmann, 1999.

[90] Koller (D.) et Parr (R.), « Policy iteration for factored MDPs », dans *Proceedings of the Uncertainty in Artificial Intelligence (UAI-00)*, p. 326–334, 2000.

[91] Feng (Z.) et Hansen (E.), « Symbolic heuristic search for factored markov decision processes », dans *Proceedings 18th AAAI*, p. 455–460, Edmonton, Alberta, Canada, 2002.

[92] Hansen (E. A.) et Shlomo (Z.), « Lao* : A heuristic search algorithm that finds solutions with loops », *Artificial Intelligence*, vol. 129, 2001, p. 35–62.

[93] Bagchi (A.) et Mahanti (A.), « Admissible heuristic search in and/or graphs », *Theor. Comput. Sci.*, vol. 24, 1983, p. 207–219.

[94] Bellman (R.), « On a routing problem », *Quarterly of Applied Mathematics*, vol. 16(1), 1958, p. 87–90.

[95] Feng (Z.), Hansen (E. A.) et Zilberstein (S.), « Symbolic generalization for on-line planning », dans *Proceedings 19th UAI*, p. 209–216, Acapulco, Mexico, 2003.

[96] Barto (A. G.), Bradtke (S. J.) et Singh (S. P.), « Learning to act using real-time dynamic programming », *Artificial Intelligence*, vol. 72, n° 1-2, 1995, p. 81–138.

[97] Bonet (B.) et Geffner (H.), « Labeled rtdp : Improving the convergence of real-time dynamic programming », dans *Proceedings 13th ICAPS 2003*, p. 12–21, Trento, Italy, 2003.

[98] Buffet (O.) et Aberdeen (D.), « Robust planning with (l)rtdp », dans *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[99] Mouaddib (A.), « Progressive goal directed reasoning for real-time systems », *International Journal of Engineering Intelligent systems*, 1995.

[100] Mouaddib (A.-I.) et Zilberstein (S.), « Knowledge-based anytime computation », dans *In proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, p. 775–783, 1995.

[101] Younes (H. L. S.), Littman (M. L.), Weissman (D.) et Asmuth (J.), « The first probabilistic track of the international planning competition », *Journal of Artificial Intelligence Research*, 2005, p. to appear.

[102] McDermott (D.), « The 1998 AI planning systems competition », *AI Magazine*, vol. 21, n° 2, 2000, p. 35–55.

[103] Younes (H. L. S.) et Littman (M. L.), « Ppddl1.0 : An extension to pddl for expressing planning domains with probabilistic effects ». Rapport technique n° CMU-CS-04-167, Carnegie Mellon University, Pittsburgh, PA, 2004.

[104] McDermott (D.). « Pddl — the planning domain definition language », 1998.

[105] Feng (Z.) et Hansen (E. A.), « Symbolic heuristic search for probabilistic planning », dans *International Planning Competition of ICAPS'04*, 2004.

[106] Karabaev (E.) et Skvortsova (O.), « Fcplanner : A planning strategy for first-order mdps », dans *International Planning Competition of ICAPS'04*, 2004.

[107] Gretton (C.), Price (D.) et Thiebaux (S.), « Nmrdpp : Decision-theoretic planning with control knowledge », dans *International Planning Competition of ICAPS'04*, 2004.

[108] Yoon (S.), Fern (A.) et Givan (R.), « Learning reactive policies for probabilistic planning domains », dans *International Planning Competition of ICAPS'04*, 2004.

[109] Bonet (B.) et Geffner (H.), « mgpt : A probabilistic planner based on heuristic search », dans *International Planning Competition of ICAPS'04*, 2004.

[110] Onder (N.), Whelan (G. C.) et Li (L.), « Probapop : Probabilistic partial-order planning », dans *International Planning Competition of ICAPS'04*, 2004.

[111] Teichteil-Königsbuch (F.) et Fabiani (P.), « Probabilistic reachability analysis for structured markov decision processes », dans *International Planning Competition of ICAPS'04*, 2004.

Liste des Définitions

Liste des Théorèmes

Liste des Algorithmes

Table des figures

Liste des tableaux

Résumé

Cette thèse porte sur la planification en environnement incertain, dont un modèle classique sont les Processus Décisionnels de Markov (MDP). Nous utilisons des modèles structurés de MDP, basés sur les Diagrammes de Décision Algébriques (ADD), qui permettent de modéliser symboliquement le système décisionnel sous une forme factorisée par variables d'état. Dans une première contribution, nous proposons de réduire le nombre de variables d'état en énumérant localement le MDP symbolique, puis en appliquant des techniques de décomposition de graphes sur le sous-MDP énuméré. Le nombre de variables diminuant, l'espace d'états et les ADD sont plus petits, ce qui facilite l'optimisation du MDP. D'autre part, le problème peut être difficile à modéliser lorsque le nombre de valeurs (arité) des variables d'état est important, car les arbres de décision du modèle deviennent très grands. Ainsi, notre deuxième contribution consiste à modéliser le problème sous forme de Réseau Bayésien Dynamique Générique et Hiérarchique, où certaines variables sont des abstractions de variables de grande arité. Notre modèle générique est paramétré puis automatiquement instancié par des macro-actions définies sur le sous-espace engendré par les variables de grande arité. Nous montrons l'efficacité de notre approche hiérarchique et symbolique sur des problèmes de déplacement et de prise d'information, dont la variable de navigation a une arité généralement très grande. Les macro-actions sont des macro-déplacements locaux, définis dans des régions distinctes du sous-espace de navigation. Enfin, dans une troisième contribution, nous proposons une classe d'algorithmes symboliques et heuristiques, qui permettent d'optimiser partiellement un MDP sur un sous-espace d'états atteignables, connaissant des états initiaux possibles du processus. Nous présentons une heuristique de plus sûr chemin stochastique, qui cible la recherche des stratégies optimales sur les buts du problème. Le MDP est ensuite optimisé en alternant une phase d'expansion du sous-espace atteignable, et une phase de programmation dynamique stochastique sur le sous-espace atteignable courant. Nous proposons également une version en ligne de notre algorithme, qui optimise le MDP sur une liste de sous-buts qui augmente incrémentalement durant la mission. Nous montrons l'efficacité de notre algorithme sur des problèmes de déplacement et de prise d'information de grande taille.

Mots-clés : planification stochastique, algorithmes symboliques et heuristiques, robotique autonome

Abstract

This PhD deals with planning under uncertainty for which Markov Decision Processes (MDPs) are a classical model. We study structured models of MDPs based on Algebraic Decision Diagrams (ADDs) which allow to symbolically model the decision system in a factored fashion using state variables. Our first contribution aims at reducing the number of state variables by locally enumerating the symbolic MDP, then by applying graph decomposition techniques over the enumerated sub-MDP. The more the number of variables decreases, the smaller state space and ADDs are, and the easier the MDP optimization is made. Moreover, the problem modelling may be difficult if some state variables have a lot of possible values (arity) because the size of decision trees used in the model noticeably increases. Also, our second contribution consists in modelling the problem with a Generic Hierarchical Dynamic Bayesian Network where some variables are abstractions of large arity variables. Our generic model is parametrized and automatically instantiated by macro-actions defined on the subspace that is generated by large arity variables. We demonstrate the efficiency of our hierarchical and symbolic approach with moving and data acquisition problems whose navigation variable arity is typically very large. The macro-actions are local movings defined in distinct regions of the navigation subspace. Finally, in our third contribution, we propose a class of symbolic and heuristic algorithms shaped to partially optimize a MDP over a reachable state subspace if some possible initial states of the process are known prior to the optimization. We present a safest stochastic path heuristics which focuses the search of optimal strategies on the problem's goals. Then, the MDP is optimized by alternating a reachable subspace expansion step and a stochastic dynamic programming step over the current reachable subspace. We also propose an on-line version of our algorithm which optimizes the MDP with a list of sub-goals which gradually increases during the mission. The algorithm efficiency is shown on the basis of large moving and data acquisition problems.

Keywords: stochastic planning, symbolic heuristic algorithms, autonomous robotics

10, avenue Édouard-Belin - BP 54032 - 31055 Toulouse cedex 4 - France
Tél : 33 (0)5 62 17 80 80 - Fax : 33 0(5) 62 17 83 30 - http://www.supaero.fr
Établissement public à caractère administratif sous tutelle du Ministère de la Défense