Processus décisionnels de Markov appliqués à la planification sous incertitude

Pierre Laroche

Département de formation doctorale en informatique
UFR STMIA

École doctorale IAE + M

Processus Décisionnels de Markov appliqués à la planification sous incertitude

THÈSE

présentée et soutenue publiquement le 27 janvier 2000

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Pierre LAROCHE

Composition du jury

Rapporteurs :	Rachid Alami, Directeur de Recherche LAAS-CNRS
	René Schott, Professeur Université Henri Poincaré - Nancy 1
	Richard Washington, NASA Ames Research Center
Examinateur :	Jean Erceau, ONERA
Directeurs :	François Charpillet, Chargé de Recherche INRIA
	Jean-Paul Haton, Professeur UHP et Institut Universitaire de France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Résumé

Mots-clés: Planification, Processus Décisionnel de Markov, Agrégation d'états, Décomposition, Robotique.

Un robot mobile évoluant dans un environnement dynamique doit faire face à de nombreuses incertitudes. Les actions qu'il exécute n'ont pas toujours l'effet escompté, et ses capteurs lui renvoient des données souvent bruitées. Les Processus Décisionnels de Markov (MDP), du fait de leur adaptation à la prise en compte d'incertitudes, sont étudiés depuis quelques années dans la communauté Intelligence Artificielle. Ceux-ci permettent le calcul d'un plan plus robuste que les méthodes de planification classiques, puisque calculé en prenant en compte les diverses conséquences probables des actions. Dans le cadre de la robotique mobile, ce modèle permet également de superviser l'exécution des missions, en utilisant des fonctions probabilistes pour aider le robot à se localiser. Mais les algorithmes classiques de résolution de MDP sont complexes, ce qui les rend difficilement adaptables à des environnements de grande taille, tels que ceux nécessités dans le cadre de la robotique.

En conséquence, beaucoup de travaux s'intéressent aux techniques d'approximation, qui permettent d'obtenir des plans sous-optimaux en un temps de calcul réduit. C'est dans ce cadre que s'inscrit cette thèse. Après avoir montré comment nous paramétrons un MDP pour l'appliquer à la robotique mobile, nous présentons deux techniques d'approximation. La première utilise l'agrégation d'états, chaque couloir de l'environnement donnant lieu à un ou plusieurs états abstraits. La seconde est fondée sur la décomposition d'environnements, lesquels sont représentés à l'aide d'un graphe valué permettant d'approcher heuristiquement les valeurs des états. Ces deux méthodes permettent de réduire considérablement les temps de calcul, et donnent des plans très proches de l'optimal.

Abstract

Keywords: Planning, Markov Decision Processes, State Aggregation, Decomposition, Robotics.

This thesis describes an application of Markov Decision Processes (MDP) to planning problems, in a mobile robotics field. An office delivery robot needs to take various sources of uncertainty in account to get robust performance. Actions are not completely reliable: for instance, due to wheel slippage, the robot can stay where it is instead of moving. The robot has similar problems with observation, because data returned by its sensors are noisy. Stochastic models such as MDPs are well suited to cope with uncertainty. Unfortunately, classical planning algorithms based on MDPs are intractable for large environments with many states. As a consequence, much research focuses on approximation

techniques, trying to reduce the search space to obtain near-optimal policies in a reduced computing time. In this thesis, we first adapt the MDP framework to mobile robotics problems, and then propose two techniques allowing to reduce the planning computing time. The first one is based on state aggregation, each corridor of the environment being treated as one or several abstract states. The second one is based on environment decomposition. Policies are approximated using a directed graph, in which each node is an intersection of the environment, and each arc is a corridor (or an office, hall, etc.) connecting two intersections. State values are computed using a heuristic valuation of the graph. These two methods allow to obtain near-optimal policies, significantly reducing computing time.

Remerciements

Cette thèse doit beaucoup à beaucoup de monde. Je vais essayer de n'oublier personne, mais c'est sans garantie...

Tout d'abord, je tiens évidemment à remercier les membres de mon jury, et tout spécialement les rapporteurs qui ont accepté de juger cette thèse. Rachid Alami, pour l'intérêt porté à mon travail et son enthousiasme. Richard Washington, pour avoir eu la gentillesse de se déplacer depuis les Etats-Unis pour assister à ma soutenance. Enfin, René Schott a énormément contribué à cette thèse : nos rencontres régulières ont toujours été très enrichissantes et ses multiples compétences (au niveau des modèles stochastiques, de la robotique mobile, de la planification de trajectoires, etc.) m'ont beaucoup aidées. Je remercie également Jean Erceau pour ses encouragements.

François Charpillet a été mon encadrant principal, et je tiens à le remercier pour sa gentillesse, sa patience et son aide durant ces trois années de thèse. Il m'a proposé un sujet passionnant, a toujours été disponible pour diriger mon travail, tout en me laissant une grande liberté à d'autres moments. Jean-Paul Haton a également encadré ce travail. Je tiens à le remercier pour l'accueil chaleureux qu'il m'a réservé lors de mon arrivée dans l'équipe RFIA. A travers lui, j'en profite également pour remercier tous les membres de cette équipe : la participation aux séminaires de l'équipe RFIA est toujours enrichissante, et la diversité des recherches effectuées dans cette équipe en ont fait un cadre de travail très stimulant.

Enfin, je tiens également à remercier Anne Boyer et Jean-François Mari, qui ont participé à mon encadrement lors de mon stage de D.E.A. J'ai régulièrement parlé de mon travail avec Jean-François pendant ces trois années ; ces discussions ont inspiré une partie de mon travail. D'autres personnes m'ont aidé plus particulièrement lors de la rédaction de ce manuscrit ou de la préparation de ma soutenance. Un grand merci à Alain Dutech et Bruno Scherrer, qui ont eu la gentillesse de relire ma thèse et de faire de nombreuses remarques intéressantes. Yann Boniface, Jean Lieber, Nicolas Rougier et l'ensemble de l'équipe MAIA m'ont quant à eux aidé à préparer la soutenance : merci !

Je remercie également mes parents et toute ma famille, ainsi que Stéphanie, qui a beaucoup été délaissée durant la rédaction de ce mémoire. Cela n'arrivera plus !

Impossible de faire une page de remerciements sans penser à tous ceux avec qui j'ai passé ces trois années. Mes collègues de bureau, pour l'ambiance en journée : Irina, Nicolas, Olivier, Marc, Arnaud et Jean. Tous les autres, pour l'ambiance de nuit, particulièrement pendant les HIBS sessions : El Moural et Peg, Jean-Luc, Manu NalyseMaster, Nonal, Rafiole et Anne-Sophie, Vanch et les moines tibétains, Olivier et Gina, Carlos et Lulu, Olivier et Angelica, Yann, Evelyne, Nico, Catherine, Franck, Alain, sans oublier ceux qui ne sont pas venus, les passants anonymes de la rue du Vieil Aître ou du rond-point du Vélodrome, les scientifiques de la Fontaine et les voisins qui n'ont pas appelé la maréchaussée.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Sujet de recherche

Dès ses débuts, l'Intelligence Artificielle (IA) s'est intéressée à la résolution de problèmes, et ces recherches ont ensuite donné lieu à une des branches de l'IA : la planification. La robotique est également un champ d'action privilégié de l'IA. Dans la mesure où il s'agit de faire effectuer à un robot une tâche habituellement dévolue à l'homme, il est utile que les robots partagent quelques capacités humaines : perception de l'environnement, acquisition de connaissances, prise de décision. Si les robots de première génération, uniquement destinés à répéter une suite de gestes au sein d'un environnement statique, n'avait que peu besoin d'« intelligence », ce besoin s'est accru au fur et à mesure de l'évolution de la robotique. Les robots actuels, dits de troisième génération, se doivent d'être mobiles, autonomes, de planifier leurs actions et de réagir aux stimuli de l'environnement. Un robot de ce type doit donc être pourvu d'actionneurs lui permettant d'agir sur son environnement, de capteurs chargés de l'observation de l'environnement et surtout être capable de prendre des décisions en fonction des situations éventuellement non prévisibles auxquelles il peut être confronté.

C'est dans ce cadre que s'inscrit l'activité Robotique de l'équipe Machine Intelligente Autonome (MAIA) autour de la plate-forme *Gaston* (figure 1.1), robot acquis par l'équipe Reconnaissance des Formes et Intelligence Artificielle (RFIA) en 1994. Ce robot, un Nomad200 commercialisé par la société Nomadics, est pourvu de différents capteurs : infrarouges, ultra-sons, caméras, auxquels s'ajoutent les capteurs de contact, utilisés en cas d'urgence. Il est composé d'une base comportant trois roues motrices et directrices, et d'une tourelle qui peut pivoter indépendamment de la base. Si notre équipe de recherche n'est pas une équipe de robotique, posséder un robot a le grand intérêt de fournir une plate-forme de tests à de nombreuses recherches, et c'est dans ce cadre que nous nous plaçons. En effet, le pilotage d'un robot mobile a rapidement orienté nos recherches vers le traitement de l'incertain : une application réelle de robotique mobile ne peut être envisagée en partant du principe que « tout se passera comme prévu ». Le robot est placé dans un environnement dynamique, dans lequel il est amené à rencontrer des obstacles mobiles ou imprévus. De plus, les capteurs dont il est pourvu sont bruités, les données odométriques qu'il fournit sont peu fiables et ses actionneurs sont sujets à de petites er-

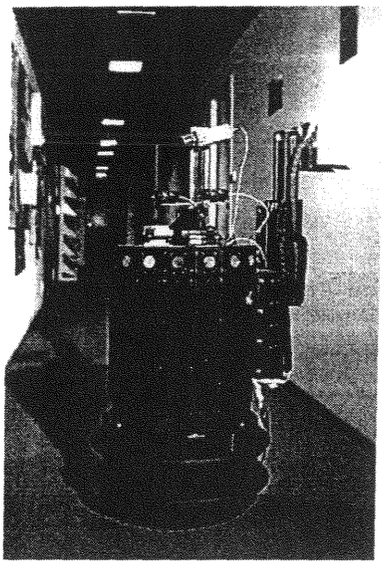


FIG. 1.1 – *Le robot Gaston.*

reurs qui, si elles ne sont pas significatives à court terme, peuvent le devenir sur un champ d'action plus étendu. Ces imprécisions introduisent un problème de localisation, qui est évidemment gênant lorsqu'il s'agit d'exécuter une action pour se déplacer vers un but.

L'acquisition de *Gaston* a coïncidé avec l'apparition d'études novatrices dans le cadre de l'IA, concernant l'utilisation d'une certaine classe de modèles stochastiques, très adaptés à la prise en compte des incertitudes : les Processus Décisionnels de Markov. Ces modèles, issus de la communauté *Recherche Opérationnelle*, étaient très peu connus dans la communauté IA avant 1994, mais depuis plus d'une dizaine de thèses y ont été consacrées, au moins en partie. Ces modèles sont surtout utilisés dans le cadre de la planification, mais sont également intéressants lorsqu'il s'agit d'apprendre les comportements d'un robot vis-à-vis de son environnement, d'aider à sa localisation, ou de modéliser les erreurs pouvant entacher ses capteurs. Malheureusement, les algorithmes classiques de planification appliqués à ces modèles sont très complexes, et les rendent peu adaptés à une application comme la robotique mobile, qui nécessite souvent la représentation explicite d'un grand nombre de paramètres. De ce fait, la majeure partie des recherches dans ce domaine s'attachent à la recherche de techniques d'approximation, permettant de calculer des plans sous-optimaux en réduisant les temps de calcul. C'est dans cette mouvance que nous nous plaçons : après avoir étudié les différents types de Processus Décisionnels de Markov pendant le stage de D.E.A., nous nous sommes rapidement tournés vers l'étude de techniques d'agrégations d'états et de décomposition d'environnements, qui constituent l'apport principal de notre travail. En effet, les environnements utilisés en robotique sont généralement représentés à l'aide d'un très grand nombre d'états, ce qui influe beaucoup sur les temps de calcul des algorithmes de planification. Nous nous sommes attachés à tirer profit des particularités de notre application à la robotique mobile pour développer diverses techniques d'approximation afin de nous approcher le plus possible d'une application temps-réel. La première technique étudiée, l'agrégation d'états, consiste à résoudre un MDP sur l'environnement global, mais en utilisant des états *abstraits* (regroupant cha-

cun un certain nombre d'états *similaires*) au lieu des états *de base*. La seconde technique, la décomposition d'environnements, cherche à résoudre le MDP global en utilisant une stratégie du type *diviser pour régner*: l'environnement global est décomposé en régions, sur lesquelles on calcule un plan, avant de réunir ces solutions partielles pour obtenir un plan utilisable sur tout l'environnement. Ces deux techniques nous ont permis d'obtenir des plans proches de l'optimal en réduisant beaucoup les temps de calcul.

1.2 Plan du mémoire

La suite de ce mémoire est organisée comme suit :

- Dans le chapitre 2, nous présentons brièvement les domaines auxquels nous nous sommes intéressés : la planification et les modèles stochastiques. Le but n'est pas de faire un historique des travaux de planification, mais simplement de montrer que de nombreux problèmes d'incertitude se posent, motivant ainsi l'utilisation des modèles stochastiques, que nous présentons ensuite. Nous commençons par les Modèles de Markov Cachés, puis nous décrivons une généralisation de ces modèles : les Processus Décisionnels de Markov Partiellement Observés (POMDP). Enfin, après avoir montré la complexité des algorithmes utilisés pour planifier dans le cadre des POMDP, nous présentons le modèle qui est le centre d'intérêt de ce mémoire : les Processus Décisionnels de Markov Parfaitement Observés (MDP). L'état de l'art est présenté dans ce chapitre puis complété en début des chapitres suivants, à chaque fois qu'une nouvelle notion est introduite.
- Le chapitre 3 est consacré à la mise en œuvre d'un MDP dans le cadre de la robotique mobile. Nous commençons par présenter les travaux combinant robotique mobile et MDP, puis nous introduisons notre approche. Nous montrons tout d'abord la structure de MDP que nous avons choisie (états représentant l'environnement, actions, traitement des obstacles, etc.), puis nous illustrons sur un exemple le type de plans (appelés *politiques* dans la littérature consacrée aux MDP) que nous obtenons en utilisant notre modèle. Puis nous justifions notre choix de l'algorithme de résolution, avant de montrer comment celui-ci peut être facilement accéléré grâce à une initialisation astucieuce. Nous présentons les résultats obtenus sur trois environnements distincts, qui seront réutilisés dans les chapitres suivants afin de comparer les résultats des différentes techniques d'approximation développées. Enfin, les deux derniers paragraphes traitent de l'exécution des politiques. Ce chapitre nous permet en outre de montrer que la représentation classique de l'environnement du robot, à l'aide d'un grand nombre d'états, influe beaucoup sur l'efficacité des algorithmes classiques de résolution de MDP.
- Dans le chapitre 4, nous présentons les techniques d'agrégation d'états que nous avons développées dans le but d'obtenir de « bonnes » politiques sous-optimales en un temps de calcul plus raisonnable que celui induit par l'utilisation des algorithmes classiques de résolution de MDP. Nous commençons par présenter les travaux existants concernant les techniques d'agrégation, puis nous introduisons notre approche. Celle-ci est fondée sur la prise en compte de la structure des environnements rencontrés dans le cadre d'une application de robotique mobile. Nous présentons suc-

cessivement quatre méthodes : les trois premières sont assez classiques puisqu'elles s'attachent à obtenir des politiques proches de l'optimal, alors que la quatrième est plus originale. En effet, celle-ci tire profit de la nécessaire utilisation d'un module de navigation lors de l'exécution des politiques pour simplifier la phase de planification. Nous présentons les résultats obtenus sur les mêmes environnements que ceux utilisés au chapitre précédent. Les politiques obtenues sont de bonne qualité et les temps de calcul sont réduits.

- Le chapitre 5 est consacré à la présentation de nos travaux dans le cadre de la décomposition d'environnements. Nous considérons ce chapitre comme l'apport majeur de notre thèse. Après nous être intéressés aux approches existantes, nous introduisons nos travaux. Ceux-ci sont fondés sur la représentation de l'environnement à l'aide d'un graphe valué, qui permet de définir efficacement les coûts de passage entre régions de l'environnement. Ces coûts sont très importants puisque c'est en fonction de ceux-ci que la politique est construite. Nous proposons deux méthodes heuristiques de valuation des nœuds du graphe, qui permettent d'obtenir des politiques très proches de l'optimal en un temps de calcul considérablement réduit, comme nous le montrons sur nos environnements de test.
- Un bilan et une comparaison des techniques d'approximation développées sont proposés dans le chapitre 6. Nous reprenons les résultats obtenus par les meilleures techniques d'agrégation et de décomposition introduites, en les comparant aux performances d'un algorithme simple de type « plus court chemin » décrit au chapitre 3.
- Le dernier chapitre est consacré à la conclusion de notre mémoire ainsi qu'à la présentation des perspectives que nous aimerions développer par la suite.

Chapitre 2

Planification et modèles stochastiques

2.1 Introduction

Ce chapitre est consacré à la présentation des deux centres d'intérêt de ce mémoire. Dans un premier temps, nous rappelons brièvement ce qu'est la planification et quels sont les problèmes que cela pose. Puis, nous décrivons les modèles stochastiques auxquels nous nous sommes plus particulièrement intéressés. Nous commençons par les modèles de Markov Cachés, bien connus dans la communauté *Intelligence Artificielle*, notamment dans le domaine de la Reconnaissance Automatique de la Parole. Puis nous présentons une généralisation de ces modèles : les Processus Décisionnels de Markov Partiellement Observés. Montrant sur un exemple la complexité de ceux-ci, nous présentons rapidement les diverses approches utilisant ces techniques, avant d'introduire un cas particulier de ces modèles : les Processus Décisionnels de Markov Parfaitement Observés. Dans la mesure où ce sont ces modèles qui nous ont tout particulièrement intéressés, nous ne présentons dans ce chapitre que les bases. En début des chapitres suivants, nous verrons à chaque fois les travaux du domaine correspondant à la nouvelle notion introduite dans le chapitre.

2.2 Planification

La planification est une discipline à part entière de l'*Intelligence Artificielle*, et cela depuis une trentaine d'années. Depuis 1992, une conférence internationale y est entièrement consacrée (*Artificial Intelligence Planning Systems*), en alternance avec une conférence européenne (*European Conference on Planning*). La communauté planification s'intéresse à la construction de systèmes capables de générer automatiquement des suites d'actions. Celles-ci (appelées plans) ont pour but de faire passer l'univers de son état initial à un état satisfaisant le but préalablement fixé. Lorsqu'on s'intéresse à un problème de planification, de nombreuses questions se posent :

- De quelle *connaissance* dispose-t-on sur le domaine d'application ?
- Quelles sont les actions autorisées et comment les modéliser ?
- Quels sont les états initiaux possibles du monde ?
- Quels sont les buts à satisfaire ? Y-a-t'il des buts prioritaires ?

- Quels algorithmes utiliser pour trouver les plans?
- Comment générer des plans rapidement?
- Comment évaluer un plan et comment s'assurer qu'un plan est de « bonne » qualité?
- Comment exécuter efficacement les plans?

Cette liste n'est bien entendu pas exhaustive. Les connaissances qu'on a sur le domaine sont très importantes : les ignorer pourrait mener à la génération de plans totalement inutiles. Selon les domaines d'application, ces connaissances peuvent être évidentes et faciles à réunir (par exemple, un robot est jusqu'à preuve du contraire incapable de traverser un mur), ou au contraire nécessiter une phase longue et difficile d'acquisition auprès d'un expert du domaine.

Dès le début des recherches faites en planification, les applications se sont naturellement tournées vers la robotique. Les travaux les plus célèbres sont sans conteste ceux décrivant le planificateur STRIPS [Fikes et Nilsson, 1971]. Celui-ci permettait au robot *SHAKEY* la résolution de problèmes simples, tels que l'empilement de blocs sur une table (le fameux *monde des blocs*). Par la suite, les travaux se sont orientés vers la planification des déplacements des robots, sous le terme de *planification de trajectoires* : placé dans un environnement éventuellement inconnu, le robot a pour but d'arriver en une certaine position, en empruntant un chemin évitant les obstacles. De très nombreux travaux sont consacrés à ce problème ; on pourra consulter [Boissonnat *et al.*, 1998] pour une étude détaillée de ces approches.

L'application de la planification au domaine de la robotique a rapidement mis en lumière un des problèmes de l'approche classique de la planification : la non-prise en compte des incertitudes. En effet, les plans sont construits en partant du principe que toute action aura des effets connus à l'avance et constants, ce qui est loin d'être toujours le cas dans des applications réalistes. L'exécution de plans dans le cadre de la robotique mobile est soumise à de nombreuses incertitudes :

- Incertitude dans les effets des actions. Suivant la structure du sol, les roues d'un robot peuvent patiner et de ce fait parcourir une distance moins grande que prévue. La présence d'obstacles inconnus lors de la génération du plan peut également fortement perturber la bonne exécution du plan.
- Incertitude sur la position initiale. Il est difficile de toujours connaître précisément la position et l'orientation du robot. Si une petite erreur n'est pas grave à court terme, elle peut en revanche avoir des conséquences fâcheuses après l'exécution d'une longue séquence d'actions.
- Imprécision des données des capteurs. Dans une application réelle de robotique mobile, le robot se localise dans son environnement grâce à ses capteurs. Ceux-ci sont généralement plus ou moins bruités, et nécessitent une interprétation. On retrouve la même imprécision au niveau des données odométriques.

La nécessité de prendre en compte les incertitudes a donné lieu à la *planification probabiliste*. Dans ce domaine, on peut citer le planificateur *BURIDAN* [Kushmerick *et al.*, 1995], qui ajoute à la représentation STRIPS classique la prise en compte des effets probabilistes des actions et permet la résolution de problèmes lorsque l'état initial n'est pas connu précisément. Une solution du problème considéré est dans ce cas une suite d'actions qui conduit à un but selon une probabilité de succès supérieure à un seuil fixé *a priori*. Mais

la prise en compte des incertitudes est encore limitée et d'autres recherches se sont alors tournées vers l'utilisation de modèles stochastiques, que nous présentons au paragraphe suivant.

2.3 Les processus stochastiques

Cette thèse s'intéresse à une certaine classe de modèles stochastiques, les Processus Décisionnels de Markov. Avant d'entrer dans le vif du sujet, nous présentons rapidement un formalisme très proche et mieux connu de la communauté *Intelligence Artificielle*: les Modèles de Markov Cachés (*Hidden Markov Models*) [Rabiner, 1989].

2.3.1 Quelques définitions

Processus stochastique

Un *processus stochastique* est une famille de variables aléatoires $X(t)$ où t est un paramètre réel prenant ses valeurs dans un ensemble T. En général, T est dénombrable et représente le temps, le processus est alors dit *discret*.

Suite stochastique

Soit un système pouvant se trouver dans un ensemble dénombrable d'états $S = \{s_1,s_2,...,s_N\}$. Entre les instants t et $t+1$, le système passe aléatoirement de l'état s_i à l'état s_j. Pour représenter ce processus aléatoire, on définit la variable q_t de la façon suivante: $q_t = s_i$ signifie que le système est dans l'état s_i au temps t. Par définition l'ensemble $(q_1,q_2,...q_t,...)$ est une suite stochastique à ensemble discret d'états.

Chaîne de Markov

Une suite stochastique vérifie la *propriété de Markov* si pour tout t:

$$Prob(q_t = s_i/q_{t-1} = s_j,q_{t-2} = s_k,q_{t-3} = \ldots) = Prob(q_t = s_i/q_{t-1} = s_j)$$

ce qui peut être explicité de la façon suivante: l'état du système à l'instant t dépend uniquement de l'état à l'instant $t-1$ (on parle alors de suite stochastique du premier ordre). On peut généraliser cela aux suites stochastiques d'ordre n, pour lesquelles l'état à l'instant t dépend des n précédents états (voir [Mari, 1996] ou [Aycard *et al.*, 1998] pour des applications des chaînes de Markov d'ordre 2).

Enfin, une chaîne de Markov est dite *stationnaire* si la probabilité de transition entre états est indépendante du temps, plus formellement si pour tout t et k:

$$Prob(q_t = s_i/q_{t-1} = s_j) = Prob(q_{t+k} = s_i/q_{t+k-1} = s_j)$$

2.3.2 Les modèles de Markov cachés

Généralement, lorsqu'on utilise un modèle markovien, chaque état correspond à un phénomène physique qui n'est pas toujours observé fiablement. On a alors une partie visible du modèle, ce que le processus a observé (par exemple un spectre de parole), et une partie cachée constituée par les états du modèle (représentant par exemple les phonèmes prononcés). L'état courant du processus est donc défini en fonction de l'observation faite (notée o_t) et de la connaissance (imparfaite) de l'état précédent. On parle dans ce cas de Modèle de Markov Caché (MMC), dont la définition suit :

> Un *modèle de Markov caché discret du premier ordre* est une chaîne de Markov stationnaire générant un processus stochastique à deux composantes : une cachée et l'autre observable.

Un MMC est alors défini par les éléments suivants :

1. Un ensemble S de N états ;
2. Une matrice de probabilités de transitions A, qui définit la topologie du modèle, c'est-à-dire quelles sont les transitions autorisées et celles qui sont interdites ;

$$a_{ij} = Prob(q_t = s_j / q_{t-1} = s_i)$$

3. Un ensemble $V = (v_1, v_2, \ldots, v_M)$ de M observations ;
4. Une matrice de probabilités d'observation B, qui définit les probabilités d'observation dans chaque état ;

$$b_i(k) = Prob(o_t = v_k / q_t = s_i)$$

5. La distribution de probabilités initiales π, qui permet de définir quels sont les états dans lesquels le processus peut se trouver à l'instant $t = 0$.

Pour représenter un MMC, on utilise généralement la notation $\lambda = \{S, A, \pi, B\}$. La figure 2.1 montre deux types de modèles de Markov : un modèle gauche-droite, dans lequel les transitions entre états ne peuvent se faire que des états de gauche vers les états de droite, et un modèle ergodique, dans lequel chaque état est relié à tous les autres états.

2.3.3 Utilisation des modèles de Markov

Les modèles de Markov Cachés sont beaucoup utilisés dans le domaine de l'apprentissage, par exemple en Reconnaissance Automatique de la Parole. Dans ce paragraphe notre but n'est pas de détailler les algorithmes fréquemment utilisés dans ce domaine. Nous nous limiterons à une description succincte, fondée sur un exemple tiré de la robotique mobile tel que décrit dans [Aycard *et al.*, 1998]. Cet exemple est centré sur le problème de la localisation d'un robot mobile évoluant dans un environnement intérieur structuré. La localisation est faite en utilisant les balises naturelles de l'environnement (couloir, intersection, porte ouverte, etc.) identifiées à l'aide de modèles de Markov : chaque balise est

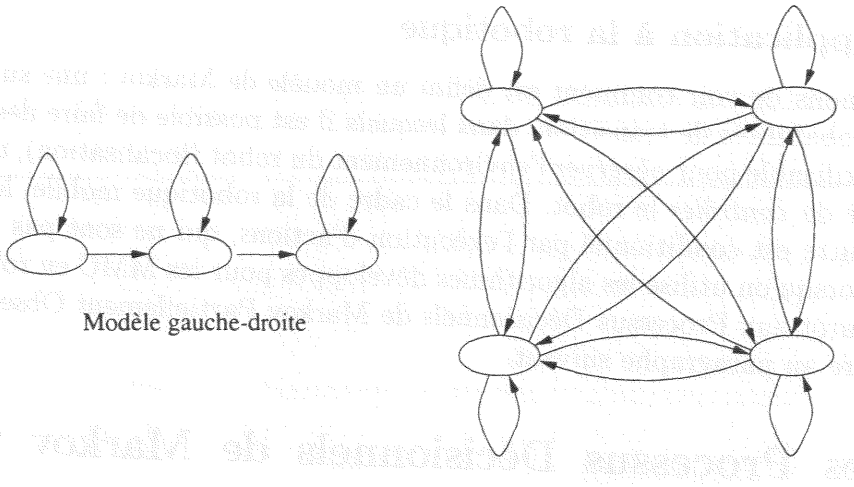


FIG. 2.1 – *Exemples de modèles de Markov.*

représentée par un modèle. Cet exemple nous permet de montrer l'intérêt de l'utilisation de tels modèles :

- Evaluation de modèle : lors des missions du robot, les modèles chargés de représenter chaque balise sont mis en concurrence, afin de déterminer lequel correspond le mieux à la séquence d'observation courante. Pour chaque modèle λ_i est calculée la probabilité $Prob(O/\lambda_i)$ que ce modèle ait engendré la suite d'observation O. Si cette suite contient n observations, cette probabilité peut être calculée en additionnant les probabilités d'observations sur tous les chemins de longueurs n : la complexité est exponentielle. Pour éviter cela, les algorithmes *forward* et *backward* permettent d'effectuer le même calcul sans énumérer tous les chemins possibles.

- Recherche du chemin le plus probable : soit une suite d'observations O et un modèle λ, comment trouver une suite d'états Q qui soit optimale selon un critère prédéfini ? Le but est ici de déterminer la composante cachée du processus en fonction des observations. Cela permet, par exemple, de déterminer les positions successives (les plus probables) du robot sachant les observations qu'il a faites. C'est généralement l'algorithme de *Viterbi* qui est utilisé pour résoudre ce problème.

- Apprentissage : pour que l'évaluation des modèles en fonction des observations faites soit efficace, il faut que chaque MMC représente fidèlement le phénomène physique (ici la balise) qu'il modélise. L'algorithme de *Baum-Welch* est généralement utilisé dans ce but : à partir d'un modèle initial λ et d'un ensemble de suites d'observations O (appelé corpus d'apprentissage), les paramètres de ce modèle sont ajustés afin de maximiser la probabilité d'observation $Prob(O/\lambda)$ du phénomène physique. Cet algorithme utilise également les fonctions *forward* et *backward*.

2.3.4 Application à la robotique

Nous venons de voir comment est défini un modèle de Markov : une suite d'états liés selon des probabilités de transition, dans lesquels il est possible de faire des observations. Si cela est utilisable pour *observer* l'environnement du robot (localisation), il nous manque ici la faculté de *contrôler* le robot. Dans le cadre de la robotique mobile, le passage d'un état à un autre est conditionné par l'exécution d'actions, qui ne sont pas modélisées ici. De ce fait, lorsqu'on utilise les algorithmes développés pour les MMC en robotique, on les applique plutôt aux Processus Décisionnels de Markov Partiellement Observés, que nous allons décrire au paragraphe suivant.

2.4 Les Processus Décisionnels de Markov Partiellement Observés

Un Processus Décisionnel de Markov Partiellement Observé (POMDP selon l'acronyme anglais) peut être vu comme un modèle de Markov Caché dans lequel les transitions entre états sont la conséquence de l'exécution d'une action. On retrouve les éléments des MMC, mais ici nous utiliserons les notations généralement utilisées dans la littérature propre au domaine. Ces modèles, issus de la communauté *Recherche Opérationnelle*, ont été le sujet de nombreuses études. On peut citer en particulier [Monahan, 1982; Lovejoy, 1991], qui proposent une vue d'ensemble des applications de ce modèle et des algorithmes de résolution couramment utilisés. Dans la suite de ce paragraphe, nous commençons par décrire formellement le modèle, puis nous montrons sur un exemple comment il peut être utilisé. Ensuite, nous présenterons diverses méthodes de calcul de plans (appelés *politiques*), avant de conclure sur ce modèle.

2.4.1 Présentation du modèle

Un Processus Décisionnel de Markov Partiellement Observé est défini par un tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, O \rangle$:

- $\mathcal{S}$: ensemble fini d'états de l'environnement ;
- $\mathcal{A}$: ensemble fini d'actions ;
- T : fonction de transition entre états, chargée de représenter les incertitudes des effets des actions ;

  $$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$$

  On note $T(s,a,s')$ la probabilité de passer de l'état s à l'état s' suite à l'exécution de l'action a.
- R : fonction de récompense qui permet de fixer le but à atteindre et les éventuelles zones dangereuses de l'environnement (états proches d'un escalier, etc.). Cette fonction peut être définie de différentes manières, suivant le problème à résoudre. On peut par exemple récompenser le fait de se trouver dans un état : on utilise alors une fonction de la forme $R : \mathcal{S} \rightarrow \mathbb{R}$. Ou alors, quand le problème étudié le nécessite, on peut être plus précis, en récompensant par exemple le fait d'être passé avec succès

d'un état à un autre en ayant exécuté une action. Dans ce cas, on utilise une fonction plus complexe $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$.

- $\mathcal{O}$: ensemble fini de symboles observables ;
- O : fonction d'observation qui spécifie la probabilité d'observer un symbole de $\mathcal{O}$ connaissant l'état dans lequel se trouve le système. Celle-ci peut être indépendante ou dépendante de l'action effectuée ; selon le cas, la fonction est définie par $O : \mathcal{S} \times \mathcal{O} \rightarrow [0,1]$ ou par $O : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow [0,1]$. On notera dans le second cas $O(s,a,o)$ la probabilité d'observer o lorsque l'action a a été exécutée dans l'état s.

Dans le cadre d'une application réaliste de robotique mobile, ce modèle est très intéressant :

- Le robot a rarement une connaissance précise de son environnement : il l'observe grâce à ses capteurs, mais ceux-ci sont bruités. L'ensemble $\mathcal{O}$ et la fonction O permettent de modéliser cela.
- Ses déplacements sont conditionnés par les actions qu'il doit exécuter, mais celles-ci peuvent avoir des effets divers selon la structure du sol (patinage des roues, présence d'obstacles imprévus, etc.) La fonction T représente ces incertitudes.
- Généralement, un robot ne se déplace pas dans son environnement sans raison : il a un but à atteindre, qui sera spécifié à l'aide de la fonction R. L'intérêt des POMDP est qu'ils permettent le calcul d'une politique qui cherchera à maximiser la récompense totale ou la récompense moyenne, selon le critère d'optimalité retenu.

Par rapport aux MMC, l'ajout des actions et de la fonction de gain a l'intérêt majeur de permettre la planification des déplacements du robot. Néanmoins, les MMC ne sont pas totalement abandonnés, puisque lorsqu'il s'agit d'apprendre le comportement réel du robot et ses capacités d'observations (apprentissage des fonctions T et O), on peut utiliser les algorithmes propres aux MMC, qu'on adapte au formalisme POMDP.

2.4.2 Exemple d'utilisation d'un POMDP

Le modèle présenté est très prometteur, mais il est également très complexe, notamment lorsqu'on s'intéresse à la planification d'actions dans un environnement partiellement observé, dans lequel l'état courant du processus n'est que partiellement connu. Généralement une politique spécifie une action optimale pour chaque état du problème, ce qui n'est pas possible dans le cadre des POMDP puisque l'état courant du processus n'est pas connu. L'idée qui vient alors immédiatement à l'esprit est de faire correspondre une action optimale non plus à un état mais à une observation. Si le robot fait telle observation alors il devra exécuter telle action. Mais on s'aperçoit vite que dans ce cas le problème n'est plus markovien : une même observation peut être faite en plusieurs états, comme on le voit sur le petit environnement de la figure 2.2, tirée de [McCallum, 1995]. A une même observation peut donc correspondre diverses actions optimales, selon la position réelle dans l'environnement. Dans ce monde à 11 états, le robot est uniquement capable de percevoir les murs directement adjacents. Les états contenant une flèche sont donc perçus de la même façon. Dans un cas, l'action optimale consiste à descendre vers le but, alors que dans les deux autres états l'action optimale est de monter. En conséquence, la

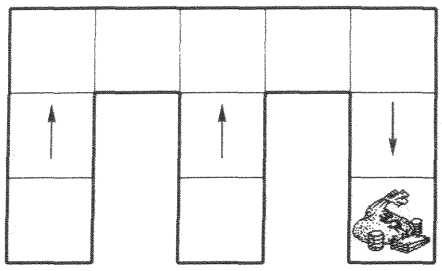


FIG. 2.2 – *une observation pour trois états → 2 actions optimales.*

plupart des travaux s'intéressant à la planification dans le cadre POMDP utilisent le fait que même si l'état courant n'est pas connu, on en a toujours au moins une vague idée. Le problème devient alors très complexe, comme nous le montrons dans la suite de ce paragraphe, à l'aide du problème classique du tigre, décrit par exemple dans [Cassandra, 1994].

Présentation du problème

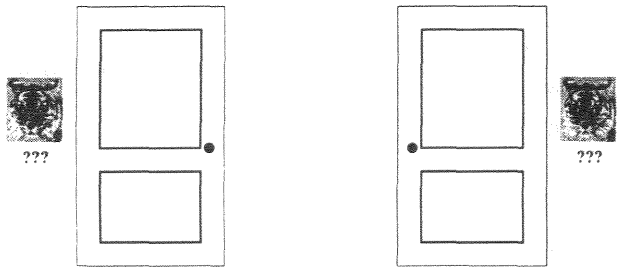


FIG. 2.3 – *Problème du tigre.*

On propose à une personne de tenter l'expérience suivante : placée dans une pièce comportant deux portes, elle a la possibilité en ouvrant une des portes, soit de découvrir un trésor, soit de se retrouver nez à nez avec un tigre affamé. Avant d'ouvrir une des portes, la personne a la possibilité d'écouter afin d'essayer d'entendre derrière quelle porte se trouve le tigre, mais à chaque fois qu'elle tend l'oreille, le trésor est amputé de quelques pièces... De plus, comme les portes sont très rapprochées, entendre le tigre derrière la porte de gauche ne signifie pas forcément qu'il est effectivement derrière cette porte. On estime à 85% la fiabilité de ces observations. La règle du jeu spécifie en outre qu'une fois une porte ouverte, la personne reçoit sa récompense ou sa punition, et le processus reprend au départ. Ces données étant fournies, comment cette personne peut-elle faire pour minimiser les chances de se faire manger tout en maximisant son gain ?

Experte en POMDP, cette personne décide de résoudre le problème en spécifiant les paramètres suivants :

- Ensemble d'états $\mathcal{S}$:
  - **TG** : le tigre est derrière la porte de **gauche**
  - **TD** : le tigre est derrière la porte de **droite**

- Ensembles d'actions $\mathcal{A}$:
  - **E** : Ecouter
  - **OG** : Ouvrir la porte de **gauche**
  - **OD** : Ouvrir la porte de **droite**
- Ensemble d'observations $\mathcal{O}$:
  - **EG** : J'entends le tigre derrière la porte de **gauche**
  - **ED** : J'entends le tigre derrière la porte de **droite**

Les tables 2.1, 2.2 et 2.3 définissent respectivement les probabilités de transition, d'observation et les gains obtenus selon l'état du processus et l'action effectuée.

T(s,E,s')	s'=TG	s'=TD
s=TG	1	0
s=TD	0	1
T(s,OG,s')	s'=TG	s'=TD
s=TG	0,5	0,5
s=TD	0,5	0,5
T(s,OD,s')	s'=TG	s'=TD
s=TG	0,5	0,5
s=TD	0,5	0,5

TAB. 2.1 – *Probabilités de transition (état, action → état).*

O(s,E,o)	o=EG	o=ED
s=TG	0,85	0,15
s=TD	0,15	0,85
O(s,OG,o)	o=EG	o=ED
s=TG	0,5	0,5
s=TD	0,5	0,5
O(s,OD,o)	o=EG	o=ED
s=TG	0,5	0,5
s=TD	0,5	0,5

TAB. 2.2 – *Probabilités d'observation état,action → observation.*

R(s,a)	a=E	a=OG	a=OD
s=TG	-1	-100	+10
s=TD	-1	+10	-100

TAB. 2.3 – *Définition des gains.*

Il est clair ici qu'un plan spécifiant une action à chaque état est aussi évident qu'inutile, puisque l'état du processus est inconnu. Voyons alors comment le problème peut être résolu.

Nécessité d'un nouvel espace d'états

La solution du problème est en apparence simple : il s'agit d'ouvrir la porte droite quand le tigre se trouve à gauche et inversement, mais la personne ne détient pas cette connaissance, elle doit donc écouter pour se faire une idée de l'état courant du processus. En fonction des probabilités d'observation, combien de fois la personne doit-elle écouter avant d'estimer avoir une connaissance suffisamment sûre de l'état du processus pour prendre le risque d'ouvrir une porte? La réponse va nous être donnée par la solution du POMDP. Ici la politique à suivre ne peut pas être un ensemble de couples *état* $\rightarrow$ *action* ou *observation* $\rightarrow$ *action*, nous l'avons déjà montré. Puisque la décision sera prise en fonction du niveau d'incertitude sur l'état courant (la position du tigre), la solution généralement utilisée est de remplacer l'espace d'états initial par un ensemble d'états qui reflète l'incertitude. On raisonne alors sur des « ensembles d'états probables » (en anglais *belief state*) [Smallwood et Sondik, 1973]. L'espace d'états considéré devient l'ensemble $\mathcal{B}$ des distributions de probabilités sur les états. Un état de notre processus est par exemple {`TG : 0,9 ; TD : 0,1`}. La solution optimale du problème affecte alors une action à chaque élément de $\mathcal{B}$. Par exemple, sur notre problème à deux états, une des politiques possibles est illustrée par la figure 2.4 : l'action à exécuter dépend de la probabilité de l'état *Tigre à gauche*. Si la probabilité que l'état courant soit *Tigre à gauche* (resp. *droite*) est inférieure à 0,1, alors l'action exécutée est *Ouvrir la porte de gauche* (resp. *droite*). Dans tous les autres cas, l'action à exécuter est *Ecouter*. Ici la solution semble simple, mais rien ne dit que cette solution est optimale, et cela devient beaucoup plus complexe lorsque la politique dépend des probabilités de plusieurs états.

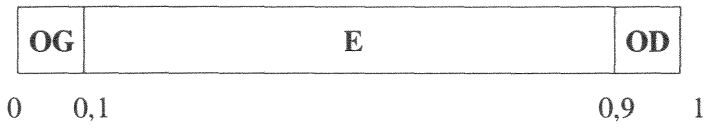


FIG. 2.4 – *Une politique pour le problème du tigre.*

La connaissance de l'ensemble d'états probables nécessite le maintien d'une fonction b, qui donne pour chaque état la probabilité qu'il soit l'état courant :

$$b : \mathcal{S} \rightarrow [0,1]$$

Cet ensemble est mis à jour après chaque action en fonction de l'ensemble d'états probables précédent, de la dernière action et de la dernière observation faite, à l'aide d'un estimateur d'état que nous appellerons EA :

$$\begin{aligned} EA_{s'}(b,a,o) &= Prob(s'/a,o,b) \\ &= \frac{Prob(o/s',a,b)Prob(s'/a,b)}{Prob(o/a,b)} \end{aligned}$$

$$= \frac{O(s',a,o)\sum_{s\in\mathcal{S}} b(s)T(s,a,s')}{Prob(o/a,b)}$$

où $Prob(o/a,b)$ permet de normaliser la somme :

$$Prob(o/a,b) = \sum_{s'\in\mathcal{S}} O(s',a,o) \sum_{s\in\mathcal{S}} b(s)T(s,a,s')$$

Comment associer une action à une distribution de probabilités ?

L'ensemble $\mathcal{B}$ de tous les ensembles d'états probables étant infini, l'espace à considérer pour construire la politique devient *continu*, ce qui complique singulièrement le problème. Comment représenter les politiques ? Comment partitionner l'espace d'états pour obtenir une politique optimale ?

Avant de répondre à ces questions, nous allons d'abord voir comment calculer une politique en faisant abstraction du caractère continu de l'espace d'états. C'est ici qu'intervient la fonction de récompense R. L'équation de Bellman [Bellman, 1957] nous donne la valeur d'un état s dans lequel on exécute l'action a :

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

Dans cette équation, le facteur γ permet de pondérer l'effet des actions à long terme par rapport à la récompense immédiate. Nous reviendrons sur l'importance de ce facteur plus tard. Dans le cadre de notre problème, l'état courant n'est pas connu, on calcule donc la valeur d'une distribution de probabilité b :

$$V^a(b) = \rho(b,a) + \gamma \sum_{b'\in\mathcal{B}} \tau(b,a,b')V(b') \tag{2.1}$$

où ρ est la nouvelle fonction de gain prenant en compte tous les états probables :

$$\rho(b,a) = \sum_{s\in\mathcal{S}} b(s)R(s,a)$$

et τ est la fonction de transition entre distributions de probabilités :

$$\tau(b,a,b') = \sum_{\{o\in\mathcal{O}|EA(b,a,o)=b'\}} Prob(o/a,b)$$

La politique optimale est obtenue en calculant pour chaque distribution de probabilité l'action optimale, c'est-à-dire l'action qui maximise la valeur de l'état donnée par l'équation 2.1. La valeur de chaque distribution étant calculée en fonction de la valeur des autres distributions, ce calcul est fait itérativement, à l'aide d'une adaptation de l'algorithme *Value Iteration* [Bellman, 1957] (voir algorithme 1).

Partition de l'espace d'états

L'algorithme présenté permet de calculer la valeur de chaque distribution de probabilités, mais il subsiste toujours un gros problème : l'espace des distributions de probabilités

Algorithme 1 Algorithme général de calcul d'une politique optimale.

Initialisation préalable
$t = 0$
pour tout $b \in \mathcal{B}$ **faire**
  $V_t(b) = 0$
fin pour

Calcul itératif des valeurs des états
répéter
  $t = t + 1$
  pour tout $b \in \mathcal{B}$ **faire**
    $V_t(b) = max_{a \in \mathcal{A}}[\rho(b,a) + \gamma \sum_{b' \in \mathcal{B}} \tau(b,a,b') V_{t-1}(b')]$
  fin pour
jusqu'à $|V_t(b) - V_{t-1}(b)| < \epsilon \ \forall b \in \mathcal{B}$

étant continu, comment calculer pratiquement ces valeurs? Une propriété importante, établie dans [Smallwood et Sondik, 1973], permet de résoudre ce problème : *sous certaines conditions* (lorsque l'horizon est fini notamment), *la fonction de valeur est une fonction convexe linéaire par parties.* Une fonction de valeur qui correspond à la politique présentée figure 2.4 est donnée par la figure 2.5.

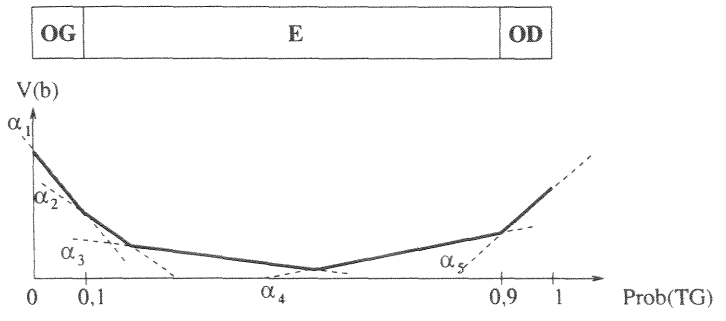


FIG. 2.5 – *Fonction de valeur convexe linéaire par parties.*

L'espace d'états est partitionné à l'aide de vecteurs à $|\mathcal{S}|$ dimensions, un vecteur donnant la valeur de chaque état selon sa probabilité. Sur notre exemple, cela est encore relativement simple puisque nous n'avons besoin que d'un espace à deux dimensions, mais cela se complique grandement lorsqu'on ajoute des états. A chaque vecteur correspond une action optimale, et le vecteur choisi pour représenter une région de l'espace d'états est celui qui maximise la valeur cumulée des états. Ainsi, la valeur d'une distribution de probabilités sur les états peut être exprimée de la façon suivante :

$$V_t(b) = \max_{\alpha \in \mathcal{C}_t} \sum_s b(s).\alpha(s) \tag{2.2}$$

où $\mathcal{C}_t$ est un ensemble *fini* de vecteurs à $|\mathcal{S}|$ dimensions. Le problème initial est alors déplacé : il s'agit maintenant de trouver ces vecteurs permettant de calculer la fonction de valeur optimale. De nombreux algorithmes ont été développés dans ce sens [Cassandra, 1998]. La plupart sont fondés sur des techniques de programmation dynamique, ce qui les rend complexes et très lents. Lorsque nous avons commencé notre étude, l'algorithme *Witness* [Littman, 1994] était le plus rapide et nous l'avons décrit en détail dans [Laroche, 1995]. Nous le présentons brièvement au paragraphe suivant, afin de montrer la complexité des mécanismes mis en œuvre.

2.4.3 Un algorithme de résolution

Une politique est représentée ici par un arbre. La racine d'un arbre correspond à un état initial possible, à chaque nœud correspond une action et chaque branche est fonction d'une observation. Les arbres sont construits incrémentalement : à l'étape t, on crée un ou plusieurs arbres de hauteur t dont la racine est une des actions possibles et les fils des arbres créés à l'étape $t-1$. A titre d'exemple, les arbres construits pour le problème du tigre à l'étape 2 sont montrés figure 2.6. Chaque arbre correspond à un intervalle de probabilité sur l'état TG. Intuitivement, le premier arbre peut être interprété de la façon suivante :
`Si la probabilité que le tigre soit à gauche est supérieure ou égale à 0,98, alors` ***`écouter`***`, puis, quelle que soit l'observation,` ***`ouvrir la porte de droite`***.
(A ce stade, certains arbres de la politique ne sont pas très efficaces : l'horizon considéré est trop faible pour obtenir une politique intéressante).

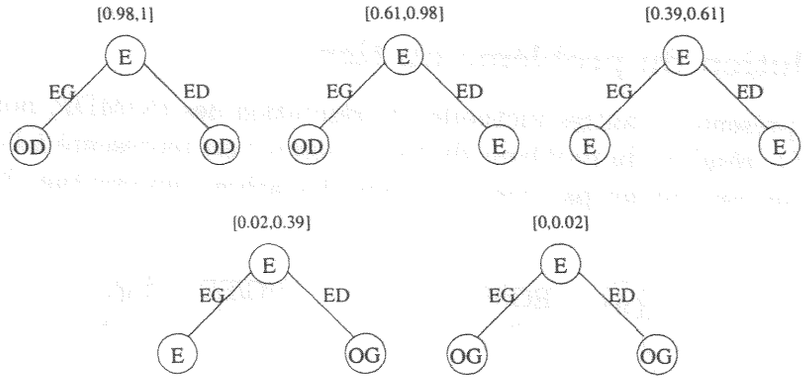


FIG. 2.6 – *Les 5 arbres de décision pour l'étape 2.*

Chaque arbre a son utilité selon la distribution de probabilité initiale. Celle-ci peut être calculée aisément en utilisant l'équation de *Bellman*, puis stockée dans un vecteur à $|\mathcal{S}|$ dimensions, une par état : le vecteur α de l'équation 2.2. La résolution du problème du tigre revient alors à trouver les arbres optimaux. Si on construit tous les arbres possibles à l'horizon t, on est sûr d'obtenir une politique optimale sur cet horizon : selon la distribution de probabilités initiale sur les états, on choisit un des arbres, puis on parcourt ses branches en fonctions des observations faites. Pour avoir une politique optimale sur un horizon infini, on itère le processus de construction d'arbre jusqu'à ce que la variation de

valeur de la politique obtenue ne varie plus que d'un ϵ. L'ensemble des arbres à l'horizon t est fini (puisque les ensembles d'actions et d'observations sont finis), mais la recherche exhaustive de tous les arbres, même pour un petit problème comme le nôtre, devient vite coûteuse, aussi bien en taille mémoire qu'en temps d'exécution. L'intérêt de *Witness* est de construire les arbres incrémentalement, en ne construisant que les arbres nécessaires. L'originalité de l'algorithme consiste en une procédure qui détermine que l'ensemble courant d'arbres est suffisant pour représenter la politique optimale. Malheureusement, la convergence de l'algorithme est très lente, et celui-ci nécessite la résolution de très nombreux programmes linéaires, ce qui est très coûteux. Pour le problème du tigre, ce n'est qu'à l'horizon 105 que l'amélioration de la politique passe sous le seuil préalablement fixé. De ce fait, dans [Littman *et al.*, 1995a], les auteurs affirment que cet algorithme, bien qu'étant le plus performant, est à limiter à des problèmes de taille très réduite : pas plus d'une centaine d'états et d'une vingtaine d'observations. Suite à *Witness* a été développé l'algorithme *Incremental Pruning*. Ces deux algorithmes sont comparés dans [Cassandra *et al.*, 1997], montrant que le second est beaucoup plus rapide que le premier (temps de calcul divisés par 5 voire plus sur certains problèmes). Malgré cela, il faut par exemple encore plus de 900 secondes de temps CPU pour trouver une politique dans un environnement à 16 états.

Ceci montre bien que les algorithmes fondés sur les ensembles d'états probables sont beaucoup trop complexes pour être utilisés sur des problèmes réels. Nous n'avons pas donné de résultats concernant la complexité des algorithmes. Ceux-ci varient selon la structure du POMDP représenté, comme le montre l'étude très intéressante et très détaillée faite dans [Littman, 1994]. On peut la résumer en affirmant que ces problèmes sont, dans le meilleur des cas, NP-difficiles.

2.4.4 Solution du problème du tigre

Avant de présenter d'autres méthodes de résolution des POMDP, nous donnons ici à titre indicatif le résultat du problème du tigre. Celui-ci est représenté à l'aide d'un graphe (figure 2.7), qui est obtenu par concaténation des arbres représentant les politiques (de hauteur 105 !)

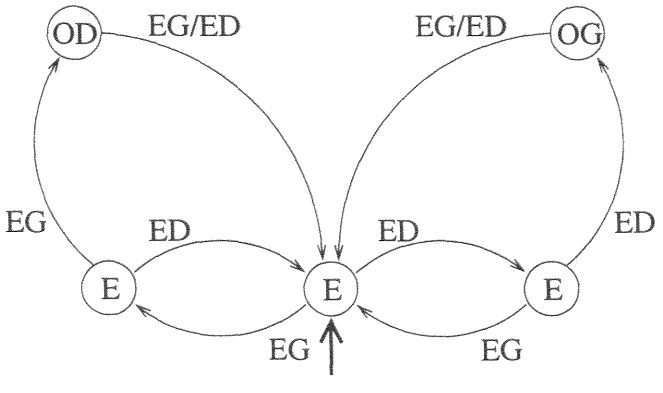


FIG. 2.7 – *Graphe solution du problème du tigre.*

Le nœud du graphe marqué par la flèche en gras est le nœud dans lequel on se trouve au

départ du processus, quelle que soit la distribution de probabilités initiale. Les transitions entre nœuds se font ensuite uniquement en fonction de l'observation faite. La politique peut être résumée ainsi : écouter jusqu'à entendre deux fois de suite le tigre derrière la même porte, puis ouvrir l'autre. Après avoir ouvert une porte, le processus est réinitialisé.

2.4.5 Autres méthodes de résolution

Nous venons de montrer sur un exemple comment sont résolus classiquement les POMDP : on ne raisonne plus sur les états, mais sur les distributions de probabilités sur ces états, d'où un espace continu qui pose beaucoup de problèmes. De ce fait, de nombreuses approches ont tenté d'introduire de nouvelles techniques pour aider à la résolution des POMDP. Dans la suite de ce paragraphe, nous présentons d'abord brièvement les travaux proches de ceux déjà présentés, puisque utilisant des distributions d'états probables, puis nous introduisons des travaux très différents, qui utilisent une mémoire du passé pour apprendre les politiques.

Utilisation d'approximations

Les travaux présentés ici cherchent à résoudre le problème de l'affectation d'une action optimale à une région de l'espace, comme c'est fait dans *Witness*.

L'approche la plus simple est décrite dans [Littman *et al.*, 1995a] : l'idée est d'interrompre *Witness* bien avant la convergence de l'algorithme, en partant du principe qu'on peut obtenir rapidement une « bonne » solution, même si elle n'est pas optimale. L'algorithme utilisé étant *anytime*, il est très simple de le contraindre sur n minutes d'exécutions ou n itérations. L'intérêt de cette approche est néanmoins très limité, puisqu'elle ne réduit en aucun cas la complexité du problème.

D'autres travaux cherchent à trouver une approximation de la fonction de valeur, et par ce biais une politique sous-optimale efficace. C'est le cas par exemple de [Parr et Russell, 1995] : la fonction convexe linéaire par parties est itérativement approchée en partant de l'hypothèse que le nombre de vecteurs nécessaires pour représenter la fonction est connu. Cette méthode améliore de façon impressionnante les temps de calcul, mais n'a pas fait l'objet d'autres articles, sans doute à cause de l'hypothèse faite, très peu réaliste.

Dans les travaux préliminaires à *Incremental Pruning* [Zhang et Liu, 1996], les auteurs partent du principe que même si l'état courant du processus n'est pas connu, on a toujours une connaissance plus ou moins grossière de la « région » de l'environnement contenant l'état courant. Cette connaissance permet de réduire l'espace de recherche, en évitant de prendre en compte des situations très peu probables.

Dans le même ordre d'idée, [Hansen, 1997] propose un algorithme capable de résoudre une classe particulière de POMDP : ceux pour lesquels l'incertitude quant à l'état courant peut être ponctuellement levée. L'hypothèse de départ est qu'une observation parfaitement discriminante peut être faite régulièrement, permettant ainsi de connaître avec certitude l'état courant. Ceci permet d'accélérer les temps de calcul et de résoudre des problèmes de plus grande taille.

D'autres travaux exploitent également la possibilité de réduire l'espace de recherche, mais en utilisant des techniques d'agrégation. Par exemple, dans [Boutilier et Poole, 1996],

des réseaux Bayésiens et des diagrammes d'influence sont utilisés pour représenter l'environnement de manière plus compacte. Ces représentations permettent de ne faire figurer que les transitions possibles, et de regrouper les états qui ont des caractéristiques communes. Dans [Helwig et Haddawy, 1996], ce sont les actions qui sont agrégées : une action abstraite peut regrouper plusieurs actions analogues, ou constituer une suite d'actions à effectuer séquentiellement. Ceci permet d'utiliser un algorithme *anytime*, qui délivre un plan d'actions abstraites qui sera raffiné au fur et à mesure de son exécution, en tenant compte des observations faites pour choisir les actions bas-niveau à effectuer.

Enfin, il est également possible de rejeter l'incertitude quant à l'état du processus lors de l'exécution de la politique, celle-ci étant construite en partant du principe que l'état courant est connu. La politique affecte alors une action à un état. Dans [Washington, 1996] par exemple, cette politique est utilisée pour trouver plus rapidement la politique optimale. Mais dans la plupart des travaux de ce type, c'est uniquement cette politique sous-optimale qui est utilisée : lors de l'exécution de la politique, l'environnement est observé, et l'action exécutée est fonction des observations faites, en utilisant par exemple l'action correspondant à l'état le plus probable. Nous reviendrons sur ces méthodes dans le chapitre 3.

Apprentissage de la politique

Pour éviter le problème de la continuité de l'espace d'états, d'autres types de recherche se sont plus focalisés sur les observations du modèle. Les approches que nous décrivons ici sont fondées sur l'apprentissage par renforcement [Kaelbling *et al.*, 1996]. Celui-ci s'applique très bien aux Processus Décisionnels de Markov lorsque l'état courant est connu : l'apprentissage est fait grâce à une exploration du monde. Cet apprentissage concerne non seulement la politique optimale, mais aussi le modèle : si la fonction de transition n'est pas connue, elle peut être apprise par expérience. Par exemple, un robot est placé dans un environnement, il exécute aléatoirement une action, et stocke la récompense ou la punition qu'il a obtenue. Il est alors dans un nouvel état, dans lequel il exécute une action, et ainsi de suite. Au bout d'un certain temps d'exploration, le robot aura appris ce qu'on appelle en anglais les *Q-values*, qui spécifient pour chaque couple (état,action) la récompense obtenue. La politique à suivre est obtenue en choisissant pour chaque état l'action qui donne la récompense la plus forte. Malheureusement, dans un POMDP l'état courant n'est pas connu, cet algorithme ne peut donc être appliqué. L'idée est alors d'apprendre la récompense moyenne pour des couples (observation,action). Mais, comme cela est très bien montré dans [Singh *et al.*, 1994], cet apprentissage peut amener à des politiques très sous-optimales. L'idée de *Singh et al.* est alors d'utiliser une politique stochastique markovienne : à une observation correspond un ensemble d'actions possibles, toutes pondérées par des probabilités. Les auteurs proposent un algorithme qui permet d'améliorer itérativement la politique [Jaakkola *et al.*, 1995], mais la convergence vers la politique optimale n'est pas prouvée.

Toujours dans le cadre de l'apprentissage, d'autres travaux ont pris le parti d'utiliser une mémoire du passé. Revenons à notre POMDP présenté figure 2.2, page 12. Cet exemple montrait que l'affectation d'une action optimale à une observation n'est pas possible, puisque pour trois états ayant la même observation, deux actions différentes sont à

exécuter pour s'approcher du but. Mais ce problème peut être résolu simplement, comme l'affirment [McCallum, 1995] et [Dutech, 1999]. Si on part du principe que les observations sont parfaitement fiables et que le processus se souvient de l'observation précédente, l'incertitude tombe. En effet, soit on vient d'un état dans lequel on a observé `Mur à l'est, Mur à l'ouest, Mur au sud` et il s'agit d'aller au nord, soit on vient d'un état dans lequel on a observé `Mur à l'est, Mur au nord`, et il s'agit d'aller au sud. L'idée générale est d'utiliser une mémoire du passé. Reste à définir la taille de cette mémoire. Dans les deux approches citées, des heuristiques sont utilisées pour déterminer quels sont les observables (suites d'observations de longueur variable) nécessaires pour représenter efficacement le problème. Ces observables sont ensuite utilisés comme états du processus, pour lesquels un algorithme classique d'apprentissage permet de déterminer les actions optimales. Ce type d'approche est très original, et permet de résoudre des problèmes de très grande taille : *McCallum* présente un problème à 21000 états et 2500 observations ! Mais ces méthodes sont coûteuses (l'espace d'états peut être très vaste) et leur adéquation à des problèmes dans lesquels les capteurs sont bruités est encore mal analysée.

2.4.6 Application à la robotique

Il est difficile de dire que les POMDP ont été appliqués à la robotique, car il n'existe pas de travaux qui se servent de ce modèle pour *planifier* les actions à exécuter. Mais certaines approches affirment combiner les deux, puisqu'elles utilisent en partie ce modèle. Nous en reparlerons au paragraphe 3.2, mais on peut citer ici les travaux les plus connus : ceux de *Koenig* et *Simmons*, autour du robot *Xavier* [Koenig et Simmons, 1998]. Si la politique est calculée par une recherche classique dans un graphe, l'exécution est elle supervisée à l'aide de fonctions de transition et d'observation (apprises grâce à une adaptation de l'algorithme *Baum-Welch*), et la localisation est faite à l'aide d'un estimateur d'état qui calcule les différents états probables.

Mais les travaux qui utiliseraient une politique calculée sur un POMDP sont encore à venir, et pour l'instant on en est encore bien loin...

2.4.7 Conclusion

De nombreuses études se sont intéressées à la résolution des Processus Décisionnels de Markov Partiellement Observés depuis ces cinq dernières années. Le fait que ceux-ci sont particulièrement adaptés à la prise en compte des multiples incertitudes qui entachent l'exécution d'un plan n'est pas un hasard. Malheureusement, si le cadre théorique est très prometteur, l'application à des problèmes de taille réaliste, comme ceux rencontrés dans le cadre de la robotique mobile par exemple, s'est révélée impossible. Tous les articles parus récemment s'accordent à dire que seuls des problèmes-jouets peuvent être traités à l'aide de POMDP.

De ce fait, nous nous sommes rapidement intéressés à un cas particulier de Processus Décisionnels de Markov, les Processus Décisionnels de Markov Parfaitement Observés (en anglais *Fully* ou *Completely Observable Markov Decision Processes*). Dans la suite, suivant ainsi les conventions en vigueur, nous les désignerons sous le terme « Processus Décisionnels de Markov ».

2.5 Les Processus Décisionnels de Markov

2.5.1 Cadre théorique

Un Processus Décisionnel de Markov (MDP) est un cas particulier de POMDP : l'état courant est à tout moment connu avec certitude, l'observation du monde n'est donc pas nécessaire. Du cadre POMDP, seuls certains éléments sont conservés :

- $\mathcal{S}$: ensemble fini d'états de l'environnement *parfaitement identifiables* ;
- $\mathcal{A}$: ensemble fini d'actions ;
- T : fonction de transition entre états selon l'action effectuée ;

$$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$$

  Comme précédemment, on note $T(s,a,s')$ la probabilité de passer de l'état s à l'état s' en effectuant l'action a.
- R : fonction de gain qui permet de déterminer le but à atteindre et les éventuelles zones dangereuses de l'environnement (états proches d'un escalier, etc.). Comme dans le cadre POMDP, cette fonction peut être définie de différentes manières, suivant le problème à résoudre. Dans la suite de ce chapitre nous utiliserons la formalisation la plus couramment utilisée : $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

L'hypothèse de complète observabilité de l'état courant peut *a priori* sembler très restrictive, mais celle-ci sera supprimée lors de l'exécution de la politique, comme nous le verrons au chapitre 3. Cette simplification a le grand avantage de permettre le calcul de politiques sur un espace d'états fini, ce qui réduit considérablement la complexité des algorithmes. Le paragraphe suivant explique ce qu'est une politique dans le cadre des MDP, et explore diverses stratégies utilisables.

2.5.2 Politiques dans le cadre des MDP

Critère d'optimalité

La politique optimale est calculée en fonction de la fonction de gain : il s'agit d'optimiser les récompenses possibles. Mais quand on parle d'optimalité, il faut d'abord déterminer si la durée de vie du processus est connue. Si oui, on se place dans un horizon fini : il faut assurer un comportement optimal durant le temps disponible. Il s'agit alors de maximiser l'espérance totale de gain, qui est exprimée ainsi :

$$E\left(\sum_{t=0}^{t=T} r_t\right)$$

Ici r_t représente la récompense obtenue à l'instant t. Mais l'horizon fini reste un cas très particulier. Dans la plupart des cas, on ne pourra pas déterminer à l'avance la durée de vie du processus. On s'intéresse donc généralement plutôt à l'horizon infini. Dans ce cas, on distingue habituellement deux critères : celui de la récompense moyenne (*average-reward*

criterion) et celui de la récompense totale pondérée (*discounted infinite horizon reward*). Ce second critère est le plus utilisé ; dans ce cas l'espérance de gain est formulée ainsi :

$$E\left(\sum_{t=0}^{\infty}\gamma^t r_t\right)$$

Ce critère est très intéressant car il permet de faire un compromis entre la récompense à court terme (produit immédiat de l'exécution d'une action), et la récompense qui pourra résulter de l'exécution de cette action, mais à plus long terme. Prenons l'exemple d'un robot placé dans l'environnement de la figure 2.8. Son but est d'atteindre le sac d'or, dans la pièce en bas à droite. Pour ce faire, partant de sa position initiale (coin supérieur gauche), il a le choix entre deux actions, qui le rapprochent toutes les deux du but :

- L'action A lui fait éviter un obstacle pour emprunter le couloir horizontal menant directement au but ;
- L'action B lui fait prendre le couloir vertical sans obstacle, mais menant au but en empruntant un très long chemin.

Dans ce cas, l'action A peut être dangereuse à court terme, mais une fois l'obstacle passé, le but sera atteint très vite. Inversement, l'action B est très intéressante à court terme puisque sans danger, mais induit un chemin beaucoup plus coûteux à long terme.

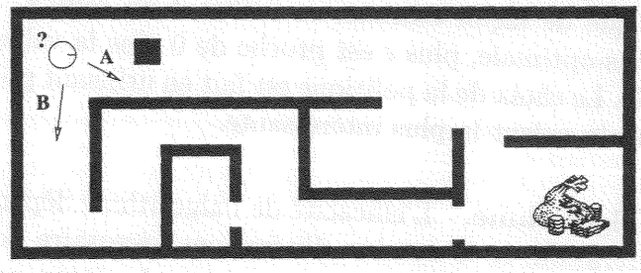


FIG. 2.8 – *Récompense à court et long terme.*

C'est le coefficient γ introduit dans le critère d'optimalité qui permet de faire ce compromis entre les récompenses immédiates et celles obtenues plus tardivement. Ce paramètre est un réel dont la valeur doit être comprise entre 0 et 1. Le faire varier permet d'obtenir différentes politiques, chacune ayant son comportement (nous en reparlerons au paragraphe 3.3.7). Dans la suite de ce mémoire, c'est donc le critère d'optimalité de récompense totale pondérée que nous utiliserons.

Forme générale

Puisque l'espace d'états est fini et que chaque état est parfaitement identifiable par rapport aux autres, une politique π affecte une action à un état ($\pi : \mathcal{S} \rightarrow \mathcal{A}$). Calculée sur un horizon infini, la politique optimale est stationnaire, c'est-à-dire qu'elle est indépendante du temps.

2.5.3 Algorithmes de calcul des politiques

Plusieurs algorithmes permettent de calculer une politique optimale dans le cadre des MDP. Ces algorithmes sont décrits notamment dans [Puterman, 1994]. Dans ce paragraphe, nous présentons les deux plus connus : *Value Iteration* [Bellman, 1957] et *Policy Iteration* [Howard, 1960].

L'algorithme *Value Iteration*

Cet algorithme très simple calcule itérativement la valeur de chaque état. La valeur d'un état est définie comme le gain obtenu par l'exécution d'une action auquel on ajoute la valeur pondérée des états qu'il est possible d'atteindre en exécutant cette même action. Plus formellement, la valeur d'un état est définie ainsi :

$$V_t(s) = max_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{t-1}(s') \right] \tag{2.3}$$

L'algorithme 2 montre comment cette valeur est calculée récursivement sur tous les états. On commence par affecter une valeur nulle à chaque état, puis on calcule successivement les valeurs de chaque état à l'horizon t, en utilisant les valeurs au temps $t-1$. Une politique à l'horizon t est optimale quand le processus s'arrête après l'exécution de t actions. Pour obtenir une politique optimale sur un horizon infini, on itère jusqu'à ce que la valeur des états ne varie plus que de façon minime, en utilisant un seuil ϵ. La fonction de valeur obtenue est alors sous-optimale, plus ϵ est proche de 0 plus la valeur des états est proche de la valeur optimale. Le choix de la politique est fait en utilisant pour chaque état l'action qui permet d'obtenir la valeur la plus intéressante.

Complexité de l'algorithme. L'efficacité de l'algorithme dépend de deux facteurs : la complexité d'une itération, et le nombre d'itérations nécessaire pour converger. Chaque itération consiste à calculer la valeur de transition entre les états, pour chaque action : cela nécessite $|\mathcal{A}||\mathcal{S}|^2$ opérations. Le nombre d'itérations nécessaire est plus difficile à déterminer. [Littman *et al.*, 1995b] montrent que l'algorithme est polynomial selon $|\mathcal{S}|$, $|\mathcal{A}|$, γ et B, où B est le nombre de bits nécessaires à la représentation des données du problème. Nous donnerons un exemple d'exécution de cet algorithme dans le paragraphe 3.4.2, page 43.

L'algorithme *Policy Iteration*

Ce deuxième algorithme est assez différent du précédent. C'est également un algorithme *anytime*, mais celui-ci calcule la politique en deux phases. Contrairement à *Value Iteration* qui calcule une politique sans connaissance *a priori*, *Policy Iteration* nécessite en entrée une politique quelconque, qui est ensuite améliorée par itérations successives. La première phase de l'algorithme consiste à évaluer la valeur de la politique courante π, en utilisant la formule suivante :

$$V_\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s,\pi(s),s') V_\pi(s') \tag{2.4}$$

Algorithme 2 *Value Iteration.*

//Initialisation des états à une valeur nulle
$t \Leftarrow 0$
pour tout $s \in \mathcal{S}$ **faire**
$V_0(s) = 0$
fin pour

//Calcul récursif des valeurs jusqu'au passage sous un seuil
répéter
$t \Leftarrow t + 1$
pour tout $s \in \mathcal{S}$ **faire**
$V_t(s) = max_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{t-1}(s')\right]$
$\pi_t(s) = argmax_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{t-1}(s')\right]$
fin pour
jusqu'à $max_s |V_t(s) - V_{t-1}(s)| < \epsilon$
retourner π_t

Calculer la valeur d'une politique revient donc à résoudre un système de $|\mathcal{S}|$ équations à $|\mathcal{S}|$ inconnues. La deuxième phase de l'algorithme, dite phase d'amélioration, cherche pour chaque état une action qui permet d'améliorer la valeur courante. L'algorithme itère jusqu'à obtenir deux politiques successives identiques, qui sont de fait optimales. Une preuve de convergence est donnée dans [Puterman, 1994].

Complexité de cet algorithme. Chaque itération de l'algorithme consiste en deux opérations : la résolution du système d'équations, qui nécessite un peu moins de $|\mathcal{S}|^3$ opérations, et la phase d'amélioration, qui est effectuée en $|\mathcal{A}||\mathcal{S}|^2$ opérations. Comme précédemment, le nombre d'itérations nécessaire à la convergence de l'algorithme est plus difficile à déterminer. [Littman *et al.*, 1995b] donnent le même résultat que pour *Value Iteration*.

Comparaison des deux algorithmes

Les deux algorithmes ont un fonctionnement très différent :

- *Value Iteration* procède par de petites améliorations successives de la fonction de valeur. En pratique, cet algorithme nécessite un grand nombre d'itérations pour converger, mais chaque itération est très rapide.
- *Policy Iteration* améliore beaucoup la fonction de valeur à chaque itération. Généralement, le nombre d'itérations nécessaire à la convergence est faible, mais chaque itération est très coûteuse.

Afin de combiner les avantages des deux algorithmes, certains travaux ont essayé de créer un algorithme hybride, présenté sous le nom de *Modified Policy Iteration* [Puterman,

Algorithme 3 *Policy Iteration.*

//Initialisation avec une politique quelconque π_i
$\pi' \Leftarrow \pi_i$

//Calcul itératif jusqu'à obtenir 2 politiques identiques
répéter
  $\pi \Leftarrow \pi'$

  //Phase d'évaluation de la politique courante
  pour tout $s \in \mathcal{S}$ **faire**
    Calculer $V_\pi(s)$ en résolvant les $|\mathcal{S}|$ équations à $|\mathcal{S}|$ inconnues suivant l'équation 2.4
  fin pour

  //Phase d'amélioration
  pour tout $s \in \mathcal{S}$ **faire**
    si il existe une action $a \in \mathcal{A}$ telle que: $\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') V_\pi(s') > V_\pi(s)$
    alors
      $\pi'(s) \Leftarrow a$
    sinon
      $\pi'(s) \Leftarrow \pi(s)$
    fin si
  fin pour
jusqu'à $\pi = \pi'$
retourner π

1994]. Mais il est encore difficile aujourd'hui de déterminer quel est l'algorithme le plus rapide. [Littman, 1996] donne différentes références, chacune affirmant la supériorité d'un algorithme par rapport à un autre. La conclusion semble être que le choix de l'algorithme doit être fait en fonction de l'application, après évaluation. C'est ainsi que nous avons procédé, comme nous le verrons au paragraphe 3.4.2.

Autres approches de résolution

Vu la complexité des algorithmes présentés, de nombreux travaux se sont penchés sur l'utilisation de méthodes permettant d'accélérer ces calculs. Nous trouvant dans ce cas, nous présenterons plutôt ces travaux ultérieurement, en début des chapitres 4 et 5. Mais une approche peut néanmoins être brièvement décrite ici, l'utilisation d'apprentissage par renforcement.

Comme nous l'avons dit au paragraphe consacré à la résolution des POMDP, ce type d'algorithmes peut être utilisé non seulement pour apprendre la politique optimale, mais aussi pour apprendre la fonction de transition si celle-ci n'est pas connue. Parmi les algorithmes d'apprentissage par renforcement, on peut citer le *Q-learning* [Watkins et Dayan, 1992] qui est le plus connu. Malheureusement ces algorithmes sont souvent très complexes et l'obtention de la politique optimale peut être très lente : la preuve de convergence de l'algorithme nécessite que chaque paire (état,action) ait été testée une infinité de fois. Certains travaux cherchent alors à décomposer les problèmes, comme par exemple dans [Singh, 1994], mais nous en reparlerons au chapitre 5.

2.5.4 Application à la robotique

Malgré l'hypothèse de départ peu réaliste dans le cadre de la robotique mobile, à savoir la parfaite observabilité de l'état courant, il existe des approches utilisant les MDP dans ce cadre. Mais comme dans le cas des POMDP, ce ne sont pas tout à fait de « vrais » MDP. En effet, si la politique est calculée à l'aide des algorithmes présentés ci-dessus, leur exécution est quant à elle supervisée par des techniques plus proches des POMDP : ensemble d'états probables, calculé grâce à une fonction d'observation. Puisque l'état courant n'est pas connu, on utilise généralement l'état le plus probable parmi ceux qui sont possibles pour déterminer l'action à exécuter. C'est dans ce cadre que nous nous plaçons, et nous décrirons quelques approches de ce type au paragraphe 3.2.

2.5.5 Conclusion

Les Processus Décisionnels de Markov, s'ils fournissent un cadre moins réaliste que les modèles prenant en compte l'incertitude de localisation, permettent en revanche de se ramener à une complexité plus abordable. Néanmoins, les algorithmes de résolution classique permettant de calculer les politiques optimales sont très dépendants du nombre d'états du processus. Or, dans le cadre de la robotique mobile, il est souvent nécessaire de représenter finement l'environnement, à l'aide de plusieurs milliers d'états. En conséquence, bien que ces algorithmes soient beaucoup moins complexes que ceux utilisés dans le cadre POMDP, il seront encore difficilement utilisables tels quels. De ce fait, nous avons

été amenés à étudier diverses méthodes d'accélération des algorithmes, que nous verrons dans les chapitres suivants.

2.6 Conclusion

Les modèles présentés ont le grand intérêt de permettre la prise en compte des diverses incertitudes auxquelles on est confronté lorsqu'on s'intéresse à la planification : problèmes d'imprécisions des commandes envoyées au robot, de bruit entachant les données des capteurs, ces deux facteurs entraînant un problème de localisation. En utilisant ces modèles, on peut espérer obtenir des plans optimaux prenant en compte le risque d'échec, et essayant de le minimiser, trouvant ainsi un compromis entre l'efficacité (la rapidité de la mission) et la sécurité, ce qui est particulièrement intéressant en robotique. Mais si le cadre théorique est très prometteur, il met en œuvre suffisamment de paramètres pour avoir à être adapté à notre application, afin d'obtenir des comportements réellement intéressants. Le chapitre suivant montre comment nous avons défini ces différents paramètres, pour que notre MDP soit en accord avec nos objectifs.

Chapitre 3

Processus Décisionnels de Markov et robotique : adéquation et difficultés

3.1 Introduction

Jusqu'à il y a encore très peu de temps, les Processus Décisionnels de Markov étaient surtout étudiés dans la communauté *Recherche Opérationnelle* et complètement ignorés par les chercheurs en Intelligence artificielle. Pourtant les Modèles de Markov Cachés, très proches des MDP, sont eux bien connus puisque intensivement utilisés en reconnaissance automatique de la parole. Ce n'est qu'au début des années 90 que la communauté IA a commencé à s'intéresser aux MDP, et cet intérêt n'est pas tombé depuis, puisqu'un article de synthèse paru dans la revue *Artificial Intelligence* y a encore été consacré très récemment [Boutilier *et al.*, 1999]. Les applications des MDP se sont rapidement tournées vers la planification d'actions, souvent en environnement intérieur, qu'il y ait derrière une réelle application en robotique mobile ou non. L'arrivée du robot Gaston dans l'équipe RFIA a suivi de peu l'émergence des modèles stochastiques appliqués à la robotique. De ce fait, nous avons été amenés à nous intéresser à ces travaux, qui, s'ils n'étaient pas nombreux à l'époque, ont engendré beaucoup d'intérêt par la suite.

Un MDP est un modèle qui peut être paramétré de diverses façons, et il y a beaucoup de choix à faire afin de passer du modèle théorique à son application. Parmi les approches existantes, très peu d'entre elles justifient ces choix, et amènent plutôt le lecteur à accepter le modèle proposé, sans se poser la question de son adéquation à l'application étudiée. La façon de représenter le monde et les buts à atteindre sont des problèmes classiques, mais un MDP doit aussi être paramétré à plus bas niveau, ce qui nécessite une bonne connaissance du modèle. L'exécution des politiques calculées est également un point à étudier, puisque comme nous l'avons vu au chapitre précédent, ce modèle est fondé sur l'hypothèse que l'état courant du processus est connu avec certitude, ce qui n'est évidemment pas réaliste dans le cadre de la robotique mobile. C'est donc à ces différents points que nous nous intéressons dans ce chapitre. Le paragraphe suivant est consacré à la description des approches robotiques utilisant des techniques stochastiques. Ensuite, nous décrivons en détail le paramétrage utilisé par notre MDP, en montrant que les politiques calculées sur ce modèle sont bien adaptées à notre application. Ensuite, nous présentons une petite étude

faite sur les deux algorithmes classiques de résolution de MDP, avant de montrer comment l'un d'entre eux peut-être accéléré par une initialisation simple. Enfin, nous exposons brièvement le mode d'exécution des politiques, puis nous concluons sur ce modèle.

3.2 Approches existantes

3.2.1 Travaux de Dean

Les travaux de *Dean* et ses collègues ont été les premiers travaux auxquels nous nous sommes intéressés. A l'époque, ceux-ci constituaient les premières recherches combinant Processus Décisionnel de Markov et robotique mobile. Dans un article paru lors d'une conférence *AAAI* [Dean *et al.*, 1993], qui a ensuite donné lieu à une parution dans la revue *Artificial Intelligence* [Dean *et al.*, 1995], les auteurs présentent une application très intéressante des MDP à la planification d'actions dans le cadre de la robotique mobile. L'environnement parfaitement observé (l'état courant est connu avec certitude) est représenté à l'aide d'une grille, chaque case de la grille représentant quatre états selon les quatre points cardinaux. Quatre actions sont possibles : une translation, un pivotement à 90° vers la droite ou la gauche, et une action de demi-tour. Le but à atteindre est une position de l'environnement. La fonction de transition est une donnée du problème (les auteurs précisent simplement qu'elle peut être déterminée empiriquement) et la fonction de gain est très simple : 0 au but, -1 partout ailleurs. Les auteurs donnent un exemple de politique montrant bien l'intérêt de l'utilisation de MDP en robotique et donnent une méthode de construction de MDP partiel permettant d'accélérer les temps de calcul de la solution optimale. Nous reviendrons plus en détail sur cet aspect de cette approche en début du chapitre suivant.

L'intérêt de ces travaux est de montrer clairement comment utiliser les MDP, quels sont leurs avantages et inconvénients, et d'ouvrir la voie à des travaux cherchant à réduire les temps de calcul de la solution optimale. Ces travaux ont éveillé l'intérêt de la communauté Intelligence Artificielle et ont été le point de départ de beaucoup de recherches. Comme nous le verrons par la suite, nous nous en sommes largement inspiré lors du paramétrage de notre modèle.

3.2.2 Le robot Dervish

Le robot Dervish, qui a gagné une des compétitions annuelles organisées par l'Association Américaine d'Intelligence Artificielle (AAAI) en 1994, n'utilise pas à proprement parler un MDP, mais s'en approche par certains points. Il n'existe pas à notre connaissance d'article vraiment technique concernant ce robot, mais une description générale se trouve dans [Nourbakhsh *et al.*, 1995]. L'environnement est modélisé selon une carte topologique, chaque bureau, couloir et intersection donnant lieu à un nœud. Ceci permet d'éviter de considérer un espace d'états trop grand : le plan est construit en prenant en compte uniquement les nœuds de la carte topologique. L'incertitude de localisation est gérée à l'aide d'un ensemble d'états probables, qui est mis à jour en fonction des perceptions du robot. L'action choisie est celle qui correspond à l'état le plus probable. De

façon assez réaliste, les états de l'environnement ne sont pas perçus parfaitement : on retrouve la fonction d'observation des MDP Partiellement Observés (appelée ici matrice de certitude) qui permet de gérer les imprécisions des capteurs du robot. L'exécution et la planification sont ici deux processus qui peuvent être appelés successivement. Un plan initial est construit entre l'état le plus probable du nœud initial et le but. Ensuite, ce plan est exécuté, jusqu'à ce que le but soit atteint ou que l'état le plus probable soit situé hors-chemin. Dans ce cas, on replanifie entre cet état et le but.

Cette approche est assez pragmatique et va à l'essentiel. Du cadre Markovien, on retrouve la fonction d'observation et l'ensemble d'états probables pour gérer les incertitudes de localisation, mais l'environnement est décrit sommairement afin de réduire le plus possible l'ensemble d'états. De plus, l'incertitude dans les effets des actions n'est pas modélisée, et la méthode de calcul du plan n'est pas donnée.

3.2.3 Le robot Xavier

Le robot Xavier de Carnegie Mellon, décrit dans [Simmons et Koenig, 1995] et [Koenig et Simmons, 1998], utilise également des techniques liées aux Processus Décisionnels de Markov Partiellement Observés. L'environnement est décrit par une carte topologique augmentée par des informations métriques. Chaque nœud du graphe est représenté par un certain nombre d'états, nombre déterminé par la longueur effective de la partie de l'environnement représentée par le nœud. Ce mode de représentation permet d'obtenir un suivi d'exécution de plan assez précis, mais a l'inconvénient de générer de gros modèles, comportant beaucoup d'états. En conséquence, le plan n'est pas calculé à l'aide d'un des algorithmes de résolution de MDP, mais par un algorithme A^*. Le suivi d'exécution de ce plan est fait de la même manière que dans le cas du robot Dervish, en maintenant un ensemble d'états probables et en considérant que le robot se trouve dans l'état le plus probable. L'originalité de cette approche vient de l'attention portée aux fonctions de transition et d'observation, qui sont apprises en utilisant une adaptation de l'algorithme *Baum-Welch* [Rabiner, 1989]. Les mécanismes d'apprentissage sont tout particulièrement décrits dans [Koenig et Simmons, 1996a] et [Koenig et Simmons, 1996b].

3.2.4 Le robot Rhino

Le robot Rhino est une plate-forme autour de laquelle se sont développées de nombreuses recherches : construction de cartes à partir des données des capteurs [Thrun *et al.*, 1998b], méthodes d'évitement d'obstacles complexes [Fox *et al.*, 1998a], localisation en environnements dynamiques [Fox *et al.*, 1998b]. Ces nombreux travaux ont permis l'utilisation de ce robot comme guide dans un musée allemand [Burgard *et al.*, 1998; Thrun *et al.*, 1998a]. La localisation du robot est faite grâce à un ensemble d'états probables, rejoignant ici Dervish et Xavier. Ce mode de localisation est amélioré grâce à un « filtre d'entropie », qui permet de ne pas tenir compte des capteurs jugés trop bruités (par la présence d'humains autour du robot par exemple). Le plan est calculé à l'aide de l'algorithme *Value Iteration*, mais peu de détails sont donnés à ce sujet, notamment au niveau du nombre d'états nécessités par l'approche.

3.2.5 Autres travaux à *Brown University*

A la suite de *Dean*, de nombreuses recherches ont été faites dans son laboratoire, recherches que nous avons déjà abondamment citées au chapitre précédent. Parmi celles-ci, un article attire plus particulièrement notre attention dans ce chapitre, puisqu'il traite de robotique [Cassandra *et al.*, 1996]. La modélisation de l'environnement ressemble à celle des travaux de *Dean*, mais ici on se place dans le cadre des MDP Partiellement Observés. Les fonctions de transition et d'observation ont été obtenues expérimentalement. Au cours de l'exécution du plan, un ensemble d'états probables similaire à celui utilisé par Dervish (appelé ici *belief state*) permet d'estimer la position courante. L'apport principal de cet article est la comparaison de stratégies : si Dervish considère que l'état courant est égal à l'état le plus probable, ici plusieurs autres possibilités sont étudiées puis testées sur plusieurs petits environnements.

Hagit Shatkay et Leslie Kaelbling, autres membres de l'université Brown, s'intéressent à l'apprentissage du modèle [Shatkay et Kaelbling, 1997; Shatkay, 1998; Shatkay, 1999]. Leur but est d'apprendre les fonctions de transition et d'observation, dans un environnement constitué par une carte topologique. Dans un premier temps, une extension d'un Modèle de Markov Caché permet d'y intégrer les données odométriques, puis une adaptation de l'algorithme Baum-Welch est utilisée pour effectuer un apprentissage rapide des deux fonctions du modèle. Cette approche est plutôt originale puisque c'est une des rares qui ne considère pas les fonctions de transition et d'observation comme des données du problème. Par rapport à l'approche de Koenig et Simmons, les modèles obtenus sont de plus petite taille, puisque la taille des nœuds de la carte topologique n'est pas explicitement représentée.

3.2.6 Conclusion

D'autres travaux dans le domaine de la robotique utilisent de près ou de loin les modèles stochastiques. Nous venons de présenter les plus importants (à nos yeux), mais on pourrait y ajouter le robot PAVLOV [Mahadevan *et al.*, 1998] ou la *BATmobile* [Forbes *et al.*, 1995].

Selon les approches, l'environnement est décrit de façon très détaillé ou au contraire réduit à une carte topologique, le plan est calculé à l'aide des algorithmes dédiés aux MDP ou selon des méthodes plus rapides, l'environnement est observé parfaitement ou partiellement, etc. Suivant les travaux de *Dean* et *Cassandra et al.*, nous avons opté pour l'utilisation d'un modèle décrivant finement l'environnement. Nous faisons l'hypothèse que celui-ci est parfaitement observé, les incertitudes de localisation étant uniquement traitées lors de l'exécution du plan. Nous décrivons en détail le modèle utilisé dans le paragraphe suivant.

3.3 Un MDP adapté à la robotique

3.3.1 Définition des états

Comme nous venons de le voir, la notion d'état peut être définie de diverses façons selon les travaux : cela peut être par exemple un ensemble de clauses représentant les connaissances que l'on a sur le monde à un instant donné [Thiébaux *et al.*, 1993; Boutilier et Dearden, 1994; Haddawy *et al.*, 1995], comme c'est souvent le cas dans les travaux classiques de planification. En robotique, la mission à remplir est généralement de se déplacer efficacement et rapidement dans un environnement, afin d'atteindre une position précise de cet environnement. De ce fait, les états définissent généralement des positions géographiques. Cela peut prendre la forme de cartes topologiques, où les états sont groupés en fonction de leur appartenance à un bureau, un couloir ou autre [Nourbakhsh *et al.*, 1995; Koenig et Simmons, 1996a; Aycard *et al.*, 1998] ou d'une grille posée sur l'environnement, chaque case de la grille constituant un ou plusieurs états [Dean *et al.*, 1993; Cassandra *et al.*, 1996; Precup et Sutton, 1997; Parr, 1998a]. Afin de permettre une localisation précise, nous avons opté pour cette dernière option : chaque case de la grille est un carré de 50 centimètres de côté (ce qui correspond à la taille de notre robot), donnant lieu à quatre états pour les quatre points cardinaux.

3.3.2 Actions utilisées

Dans la littérature consacrée à la planification, on trouve plusieurs types d'action. On a d'une part des actions de haut-niveau telles que « Chercher un café », « Eviter obstacle » ou « Traverser couloir », qui peuvent être éventuellement instanciées de différentes manières selon les circonstances. D'autre part, on a des actions plus simples et plus facilement contrôlables, telles que « Faire demi-tour » ou « Avancer d'un mètre ». Dans le cadre MDP et robotique, c'est souvent ce type d'actions qui est utilisé. C'est le niveau de précision requis pour l'exécution des politiques qui impose cela : pour gérer efficacement les incertitudes de localisation, le découpage en états est souvent très précis. En conséquence, les actions choisies sont atomiques, ce qui permet de définir simplement les fonctions de gain et de transition. Nous avons donc choisi trois actions de ce type : « Avancer », « Pivoter vers la gauche de 90° » et « Pivoter vers la droite de 90° ».

3.3.3 Fonction de gain

Comme nous l'avons déjà vu au chapitre précédent, cette fonction permet de désigner le but à atteindre et de spécifier des zones dangereuses de l'environnement, proches d'une cage d'escalier par exemple. Les états constituant une zone dangereuse se voient allouer une punition et les états contenant le (ou les) but(s) à atteindre une récompense. Mais que faire pour les états n'entrant pas dans ces deux catégories? *A priori*, plus l'état dans lequel se trouve le robot est proche du but, plus la récompense doit être forte. Cela nous donne une fonction de gain qui peut être calculée en fonction de la distance entre l'état considéré et le but. Mais si on examine la façon dont est calculée la politique optimale

permettant d'atteindre le but, on s'aperçoit que la fonction de gain peut être beaucoup plus simple.

En effet, le plan choisi est celui qui maximise la valeur de chaque état, cette valeur étant calculée grâce à l'équation de Bellman (équation 2.4, page 24), c'est-à-dire en fonction de la valeur (pondérée par la probabilité de transition entre états) de chaque état accessible à partir de l'état considéré. Si la fonction de gain affecte une récompense forte au fait de se trouver dans l'état but, les états à partir desquels le robot peut accéder directement au but auront une valeur forte. Les états adjacents à ces états auront une valeur un peu moins forte, et ainsi de suite, la valeur diminuant au fur et à mesure que l'on s'éloigne du but. Ainsi, il apparaît qu'il suffit de distinguer le gain obtenu au but des gains obtenus dans les autres états pour obtenir une politique optimale. Ceci permet de générer une fonction de gain très simple, utilisée d'ailleurs dans [Dean *et al.*, 1993] :

$$R(s) = \begin{cases} 0 & \text{si } s \text{ est l'état but} \\ -1 & \text{sinon} \end{cases}$$

Remarque 1 : afin de s'assurer que le but ait la valeur la plus forte possible (c'est-à-dire 0), il est défini comme un état absorbant : une fois le but atteint, il est impossible d'en sortir, quelle que soit l'action exécutée.

Remarque 2 : si la fonction de transition est déterministe, une fonction de gain de ce type permet d'obtenir des politiques optimales selon le critère du plus court chemin.

Cette fonction très simple est satisfaisante ; les quelques expériences que nous avons faites en essayant de donner une récompense aux états en fonction de la proximité du but n'ont montré aucune amélioration, que ce soit au niveau de la qualité des politiques obtenues ou de leur rapidité de calcul.

3.3.4 Critère d'optimalité

La fonction de gain définie ci-dessus est utilisée pour calculer une politique qui permettra d'atteindre un but fixé. Mais à une fonction de gain peut correspondre diverses stratégies optimales, en fonction du critère d'optimalité (voir le paragraphe 2.5.2, page 22). En effet, si on ne s'intéresse qu'aux récompenses immédiates, il suffit de choisir, en chaque état, l'action qui mène vers l'état le plus intéressant (c'est-à-dire le plus récompensé). Ce genre de comportement est évidemment peu efficace, il faut prendre en compte les conséquences des actions à plus long terme. Parmi les critères d'optimalité, celui qui cherche à maximiser l'espérance pondérée de gain futur (*discounted infinite horizon reward*) est le plus souvent utilisé. Celui-ci donne plus d'importance aux effets immédiats, et plus les conséquences sont lointaines, moins elles sont importantes. Ce critère est assez naturel en planification. Quand une personne se rendant à son domicile distant de dix kilomètres fait un détour de 100 mètres, ce n'est pas bien grave. En revanche, si elle se trouve sur le trottoir d'en face et que le plan suivi lui fait faire un détour de 100 mètres également, c'est beaucoup plus fâcheux. Ici dans un cas la personne est loin du but, l'effet du détour est peu important, mais dans le second cas, il a un effet néfaste immédiat. Ce critère d'optimalité permet de trouver un compromis entre les effets immédiats et les effets plus lointains, c'est donc celui-ci que nous allons utiliser par la suite.

3.3.5 Traitement des obstacles

Lors de l'exécution d'une mission, un robot mobile est confronté à plusieurs types d'obstacles. Ceux-ci peuvent être mobiles ou non, connus à l'avance ou inconnus. Si l'évitement d'obstacles mobiles et/ou inconnus ne peut être traité que lors de l'exécution des politiques, il va de soi que celles-ci doivent prévoir l'évitement des obstacles connus. Pour ce faire, deux solutions sont possibles :

1. Interdire les actions menant à une collision. Puisque l'environnement est connu lors de la construction de la politique, on peut définir pour chaque état l'ensemble des actions autorisées. Si un état est adjacent à un état obstrué, on peut interdire l'exécution d'une action de transition menant très probablement à une collision. Cette solution a l'avantage de réduire le nombre d'actions possibles par état, ce qui influe sur la complexité de l'algorithme. Mais d'un autre côté, cela nécessite un pré-traitement des états en fonction de leur environnement local. De ce fait nous n'avons pas choisi cette option.
2. Utiliser la valeur des états contenant un obstacle. En effet, puisque les politiques optimales maximisent l'espérance de gain pour chaque état, donner une valeur faible ou négative aux états obstrués permet la génération naturelle de chemins qui évitent les collisions. Pour affecter une valeur négative aux états obstrués, nous avons deux possibilités : leur affecter une punition forte (via la fonction de gain) ou les rendre absorbants. La première option possède plusieurs inconvénients :
   - la punition à affecter doit être fixée expérimentalement, selon la taille et la structure de l'environnement considéré. Une punition trop forte va générer des chemins qui s'éloignent beaucoup des obstacles, faisant ainsi des détours longs et coûteux. Inversement, une punition trop faible n'aura que peu d'effets.
   - si les obstacles ne sont pas absorbants, leur valeur sera fonction des états adjacents. Elle devra donc être calculée par l'algorithme de résolution, ce qui ajoute à la complexité, puisque le nombre d'états considérés est augmenté par tous les états obstrués.

   En revanche, la deuxième solution permet de donner une valeur très faible aux obstacles, tout en conservant la simplicité de notre fonction de gain. La valeur d'un état absorbant s est la suivante :

$$V_\pi(s) = R(s) + \gamma(1 \times V_\pi(s)) \Rightarrow V_\pi(s) = \frac{-1}{1-\gamma}$$

Comme nous le verrons ultérieurement, cette façon de traiter les obstacles permet de générer des comportements très intéressants dans le cadre de la robotique mobile.

3.3.6 Fonction de transition

La fonction de transition est un élément fondamental dans l'utilisation d'un MDP. C'est elle qui permet de gérer l'incertitude inhérente au déroulement d'une action. Dans la plupart des travaux alliant MDP et robotique [Cassandra *et al.*, 1996] [Dean *et al.*, 1993], cette fonction est considérée comme une donnée du problème, ou est générée expérimentalement, mais sans qu'on sache selon quel procédé. Afin que la fonction de transition

reflète le plus fidèlement possible le comportement de notre robot, nous en avons effectué un apprentissage en utilisant l'algorithme Baum-Welch tel que modifié dans [Koenig et Simmons, 1996a].

En fait, le comportement de notre robot est lié au navigateur [Aycard *et al.*, 1997b] utilisé pendant l'exécution du plan. Celui-ci permet d'éviter les obstacles mobiles ou non prévus lors de la phase de calcul du plan (nous en reparlerons en fin de ce chapitre). La fonction de transition dépend alors des éventuels obstacles situés autour du robot lors de l'exécution de l'action. Pour être précis, il faudrait définir une fonction de transition par configuration possible (obstacle sur la droite du robot, sur sa gauche, devant lui, etc.) En pratique, nous n'utilisons qu'une seule fonction de transition, qui « résume » les diverses situations possibles. La figure 3.1 représente les probabilités de transition apprises pour les trois actions, l'état initial étant l'état central du bas, orienté vers le nord.

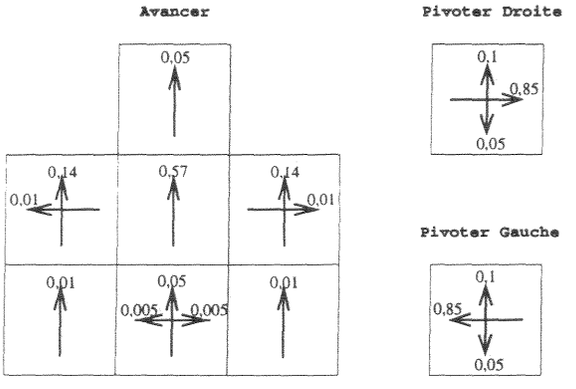


FIG. 3.1 – *La fonction de transition apprise*

3.3.7 Valeur de γ

Le paramètre γ a beaucoup d'influence. Son rôle est d'accorder plus ou moins d'importance aux gains futurs. Pour un modèle donné, de nombreuses stratégies peuvent être obtenues, en fonction de la valeur de ce paramètre. Dans chaque état du MDP, un choix est fait entre le gain à court terme (gain immédiat obtenu du fait d'exécuter une action), et le gain à plus long terme. Dans [Marion, 1996], un petit exemple amusant montre comment le facteur γ permet de choisir entre une action procurant un plaisir immédiat mais impliquant une damnation éternelle et une autre action bien ennuyeuse celle-là mais permettant d'accéder au paradis. Les politiques calculées dans le cadre de notre application sont également dépendantes de ce paramètre.

Pour illustrer cela, nous prenons deux exemples simples. Le premier est décrit par la figure 3.2. Dans ce petit environnement, il s'agit simplement d'atteindre l'état contenant le but, qui est matérialisé par une croix. Cet environnement contient 36 états *libres*: 9 positions non obstruées dans lesquelles le robot peut être orienté selon les quatre points cardinaux. A ces 36 états doivent être ajoutés les 4 états constituant le but, mais comme on ne calcule pas d'action pour ces états (puisqu'ils sont absorbants), ils n'influent pas sur la complexité des algorithmes de résolution, et de ce fait nous ne les comptons pas parmi les états libres.

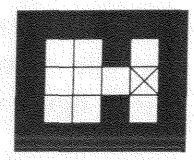


FIG. 3.2 – *Petit environnement (36 états).*

La figure 3.3 montre deux politiques optimales calculées pour cet environnement, mais en utilisant deux valeurs du paramètre γ différentes : 0,90 à gauche et 0,95 à droite. Sur chaque politique, une position de l'environnement est représentée par un cercle. Ce cercle est plein si la position est obstruée (partiellement ou non). Une action `Avancer` est représentée par une flèche reliant un cercle à un autre. Les actions de pivotement sont représentées par des flèches sur les arcs de cercle. Ces deux politiques diffèrent pour les deux positions les plus à gauche, en haut et en bas. En effet, sur la figure de gauche, l'action optimale de l'état orienté à l'est est d'avancer jusqu'à la position adjacente. En revanche, sur le figure de droite, il est préconisé de pivoter puis d'avancer vers l'état du milieu, en dessous de la position courante. Dans les deux cas, l'action choisie permet de se rapprocher du but. En revanche, avancer à partir d'une des deux positions de la figure de gauche a un risque : celui d'avancer trop, ce qui entraîne une collision avec l'obstacle situé à la deuxième colonne de la première ligne. En pivotant puis en avançant, ce risque n'existe pas, puisque même si on avance trop, on arrive dans un état libre, sans collision. Sur la figure de gauche, le paramètre γ minimise le coût de collision à deux titres, puisqu'il pondère la transition vers l'obstacle et intervient dans la valuation des états contenant un obstacle. La valeur de ce paramètre permet donc d'obtenir des politiques plus ou moins prudentes : sur la figure de droite, la politique ne prend pas le risque de collision. Dans ce cas, ce choix n'influe pas sur la distance à parcourir, mais nous verrons ultérieurement que ce n'est pas toujours vrai.

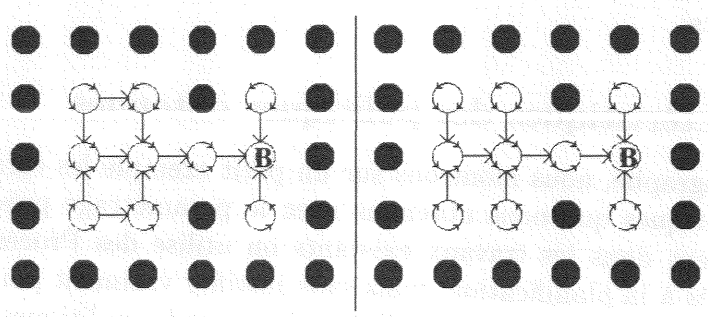


FIG. 3.3 – *Politiques optimales - γ fixé à 0,9 puis à 0,95*

La valeur de ce paramètre n'est pas fixée uniquement en fonction de la stratégie qu'on désire obtenir, mais aussi de l'environnement et de la position du but dans l'environnement. C'est ce que nous montrons sur la figure 3.4. Sur la figure la plus à gauche, γ a été fixé à 0,75, et la politique obtenue est satisfaisante. Sur la figure du milieu, nous avons simplement déplacé le but à atteindre, sans changer la valeur de γ. La politique préconisée ici n'est pas utilisable. En effet, partant de la position située en ligne 3, colonne 1, il est

aussi coûteux de se déplacer vers le but que d'entrer en collision avec les murs environnants. Dans ce cas, le but est placé trop loin de cette position pour que la récompense obtenue au but soit prise en compte. Sur la figure la plus à droite, γ a été fixé à 0,99, et la politique est correcte.

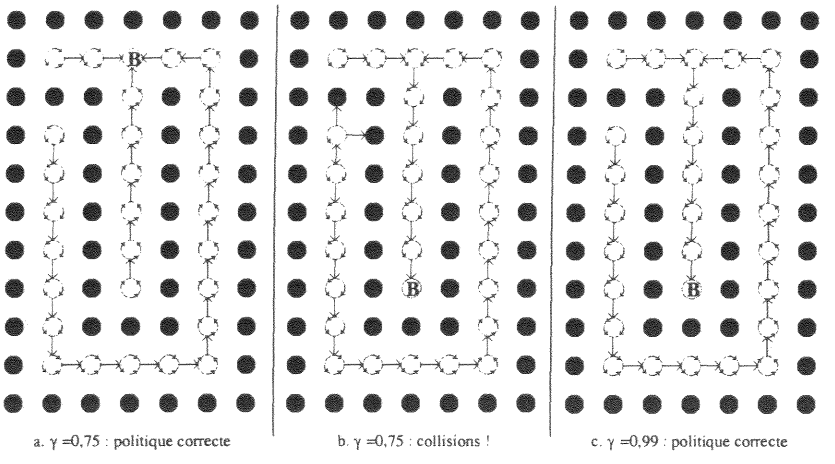


FIG. 3.4 – *Politiques optimales dans diverses configurations.*

La valeur de γ doit donc, selon la taille des environnements, être plus ou moins proche de 1. Afin d'assurer l'obtention de politiques sans collision même dans de très grands environnements, on pourrait fixer ce paramètre à une valeur très proche de 1. Le problème est que ce paramètre a aussi une influence sur la complexité des algorithmes de résolution. Plus γ est proche de 1, plus il faudra d'itérations aux algorithmes de résolution avant de converger vers la solution optimale. Il est donc nécessaire de trouver un compromis et après diverses expériences, il s'est avéré qu'une valeur de 0,99 permet d'obtenir des plans satisfaisants dans le type d'environnement que nous utilisons, même quand ils comportent plus de 4000 états.

3.3.8 Caractéristiques des politiques obtenues

Dans ce paragraphe, nous montrons sur un petit exemple les caractéristiques présentées par les politiques que nous obtenons avec le paramétrage présenté au paragraphe précédent. Souvent dans les travaux existants on utilise des Processus Décisionnels de Markov appliqués à la planification, mais sans justifier vraiment pourquoi. Or, au vu du coût des algorithmes de résolution de MDP, il faut que les politiques obtenues en vaillent la peine ! Quels sont les impératifs de notre application à la navigation d'un robot mobile ? Les missions confiées doivent être réussies, en évitant les collisions et en un minimum de temps. Le critère de distance a évidemment son importance pour assurer la rapidité d'exécution, mais dans le cas d'un robot mobile, ce critère n'est pas suffisant. En effet, selon l'environnement local du robot, la vitesse de déplacement doit être adaptée. Pour éviter les collisions, le robot ralentit dans les zones obstruées, alors qu'il peut se permettre d'accélérer s'il se trouve au milieu d'un vaste hall par exemple. Après un choc contre un mur ou un obstacle quelconque, le robot devra être repositionné afin de pouvoir continuer

sa mission en évitant la collision, ce qui peut être lent et difficile. Les collisions doivent donc être évitées autant que faire se peut. Comme nous l'avons vu au paragraphe précédent, les politiques générées par un MDP permettent de générer de tels comportements et tous les paramètres présentés précédemment ont été choisis dans ce but : les politiques doivent être à la fois sûres et efficaces. En conséquence, les chemins choisis doivent présenter un compromis selon la distance à parcourir et la sécurité du robot sur ces chemins. Un chemin court mais obstrué, dans lequel le robot devra avancer lentement pour se faufiler entre les obstacles, est dans certains cas moins intéressant qu'un chemin plus long dénué d'obstacles, dans lequel le robot pourra se déplacer rapidement et à moindre risque. Nous illustrons ceci sur le petit environnement présenté figure 3.5, dans lequel le but à atteindre est matérialisé par une croix. Cet environnement comporte un couloir adjacent au but dans lequel on a disposé deux obstacles placés en chicane, afin que le robot soit obligé de ralentir fortement pour atteindre le but. En revanche, le couloir vertical menant au but est plus large que les autres, ce qui devrait permettre au robot d'avancer plus rapidement et de façon plus sûre. La politique optimale calculée sur cet environnement est présentée figure 3.6. Un exemple de chemin est donné entre un état situé en bas à gauche et le but ; il est matérialisé sur la figure par des cercles partiellement pleins.

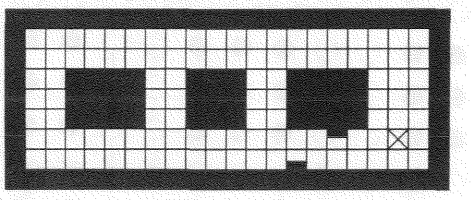


FIG. 3.5 – *Environnement simple (416 états).*

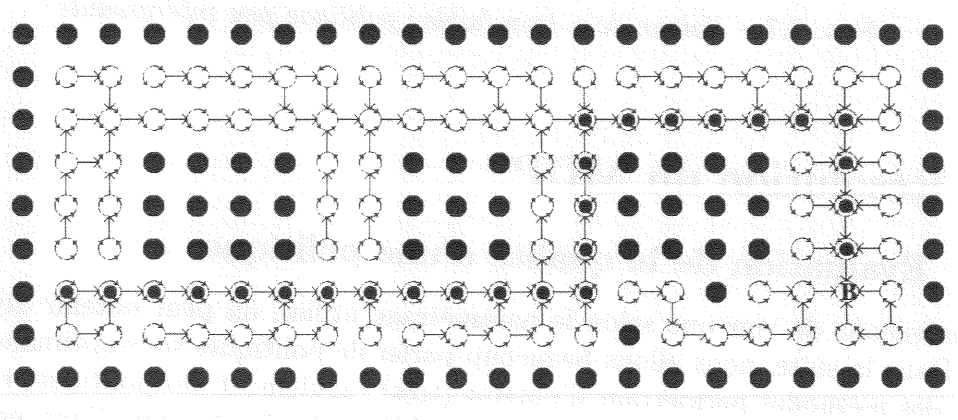


FIG. 3.6 – *Exemple de politique optimale.*

Deux comportements *a priori* intéressants dans le cadre de notre application sont générés :

- Le couloir obstrué est majoritairement évité. Un détour par le couloir du haut, qui permet d'atteindre le but via le couloir vertical le plus à droite, est préconisé par cette politique. Il n'y a que dans les états du couloir horizontal adjacent au but que le chemin à prendre passe au milieu des obstacles. Dans ce cas, le but est si proche

que le détour par le couloir du haut est plus coûteux que le risque de passage entre les obstacles.

- Dans le couloir vertical de droite, qui est plus large que les autres, la politique mène le robot dans les états centraux. En effet, longer les murs pourrait entraîner une collision, alors qu'emprunter le milieu du couloir est beaucoup plus sûr.

Ceci est particulièrement intéressant dans le cadre de notre application. Une politique générée par un MDP qui ne présenterait pas ces caractéristiques (la première surtout, car le placement au milieu des couloirs peut être pris en charge lors de l'exécution de la politique, comme nous le verrons ultérieurement) n'aurait que peu d'intérêt. Pour revenir sur l'importance de la valeur donnée au paramètre γ, la figure 3.7 présente la politique obtenue avec $\gamma = 0{,}9$. Cette valeur trop faible donne une politique très classique, dont le critère d'optimalité prend surtout en compte la distance à parcourir. L'intérêt d'utiliser un modèle stochastique pour obtenir une telle politique nous semble bien faible, d'où l'importance de choisir précautionneusement les paramètres du MDP.

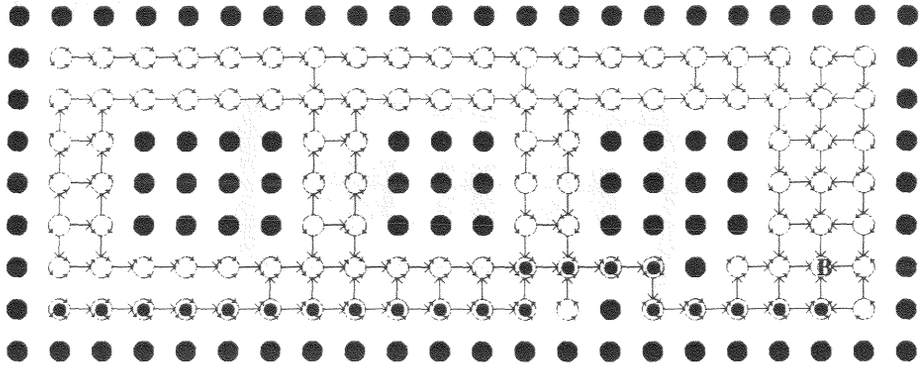


FIG. 3.7 – *Valeur de γ trop faible : politique peu intéressante.*

3.4 Résolution du MDP

3.4.1 Evaluation de la qualité d'une politique

Nous venons de voir que selon le paramétrage utilisé, on peut obtenir diverses politiques. Dans la suite, nous allons beaucoup parler de politiques sous-optimales, en introduisant des méthodes permettant d'obtenir (très) rapidement des politiques très proches des politiques optimales. Se pose alors le problème de l'évaluation d'une politique par rapport à une autre, afin de déterminer la distance vis-à-vis de l'optimal. Dans la suite de ce mémoire, nous dirons qu'un politique proche de l'optimal est une politique de bonne *qualité*.

Lorsqu'on veut évaluer la qualité d'une politique par rapport à la politique optimale et que celle-ci n'est pas connue, on utilise classiquement l'erreur de Bellman maximale (*maximum Bellman error*) [Williams et BairdIII, 1993; Parr, 1998a]. L'erreur de Bellman d'une politique π pour un état s est en fait la différence de valeur de cet état entre deux

calculs successifs de l'équation de Bellman :

$$BE(V_\pi(s)) = \max_a \left[R(s) + \gamma \sum_{s'} T(s,a,s') V_\pi(s') \right] \quad - \quad V_\pi(s)$$

[Williams et BairdIII, 1993] démontrent que l'erreur de Bellman maximale $BE(V_\pi) = \max_s BE(V_\pi(s))$ peut être utilisée pour évaluer la valeur optimale V^* en chaque état :

$$V^*(s) \leq V_\pi(s) + \frac{BE(V_\pi)}{1-\gamma}$$

Ceci est très intéressant puisque sans connaître la politique optimale, on peut obtenir une borne inférieure de la qualité d'une politique sous-optimale. Mais quand on connaît la politique optimale π^*, la qualité d'une politique peut être évaluée précisément. Attention cependant : un simple rapport V_π / V_{π^*} (avec $V_\pi = \sum_s V_\pi(s)$) va surévaluer la qualité de la politique. Pour avoir un critère d'évaluation correcte, nous utilisons $V_{\pi_{\texttt{pire}}}$ qui représente la qualité de la pire politique possible :

$$q = \frac{V_\pi - V_{\pi_{\texttt{pire}}}}{V_{\pi^*} - V_{\pi_{\texttt{pire}}}}$$

Dans notre application, la politique $\pi_{\texttt{pire}}$ correspond à une politique pour laquelle l'action optimale de chaque état est une action de pivotement, ce qui revient à rendre tous les états absorbants. De ce fait, la valeur de cette politique est très simple à calculer : $V_{\pi_{\texttt{pire}}} = |\mathcal{S}_{\texttt{libre}}| \times -1/(1-\gamma)$ où $\mathcal{S}_{\texttt{libre}}$ représente l'ensemble des états libres (états non obstrués et ne constituant pas un but).

Ce critère sera celui que nous utiliserons principalement par la suite (en employant le terme « qualité »). Mais puisqu'il utilise la fonction de valeur, le coût d'une erreur (choix d'une action différente de l'action spécifiée par la politique optimale) est plus ou moins important selon la position de l'état considéré par rapport au but : un mauvais choix d'action sur un état adjacent au but est beaucoup plus pénalisé qu'un mauvais choix sur un état situé à l'autre extrémité de l'environnement. Afin de se faire une idée supplémentaire de la qualité d'une politique, nous donnerons également un second chiffre, qui correspond tout simplement au pourcentage d'états dans lesquels l'action choisie est différente de l'action optimale. Ce critère est indépendant de la position des états vis-à-vis du but et met toutes les erreurs au même niveau.

3.4.2 Choix de l'algorithme de résolution

Les deux algorithmes classiques de résolution de Processus Décisionnel de Markov ont chacun leurs supporters : selon *Puterman* [Puterman, 1994], *Value Iteration* est plus rapide, mais *Tijms* [Tijms, 1986] favorise *Policy Iteration*. En fait, les deux algorithmes ont des fonctionnements très différents : *Value Iteration* nécessite de nombreuses mais rapides itérations, alors que *Policy Iteration* converge au bout de très peu d'itérations, mais chacune d'entre elles nécessite beaucoup de temps. *Littman* [Littman, 1996] laisse entendre que le choix de l'algorithme pourrait dépendre de la structure du MDP à résoudre, mais

ne donne pas de méthode permettant de choisir le meilleur algorithme en fonction du type de MDP. Afin d'illustrer le fonctionnement de ces algorithmes et de motiver notre choix, nous comparons les deux algorithmes sur un petit exemple à 36 états (figure 3.2, page 37) dans les paragraphes suivants.

Policy Iteration : politiques successives

Le principe de *Policy Iteration*, vu au paragraphe 2.5.3 (page 24), est très simple : partant d'une politique initiale quelconque, l'algorithme cherche à maximiser la valeur de chaque état, et itère jusqu'à obtenir deux politiques successives identiques, qui sont alors optimales. Pour effectuer la résolution du système d'équations nécessaire au calcul de la valeur de la politique courante, la librairie `LAPACK++` (*Linear Algebra PACKage in C++*)[Dongarra *et al.*, 1993] a été utilisée.

La figure 3.8 montre les politiques obtenues à chaque itération d'une exécution de l'algorithme. La politique initiale est totalement aléatoire, sans chercher à éviter les obstacles. Sa valeur (somme des valeurs des états libres) est égale à -3507 (qualité : 3,97% de l'optimal). Dès la première itération, la politique est fortement améliorée : si des collisions subsistent encore, la politique prend forme et les positions adjacentes au but se voient affecter une action qui semble optimale ; la valeur de la politique passe à -1952 (70,14% de l'optimal). Après la seconde itération plus aucune collision ne subsiste et la politique est quasiment optimale (valeur : -1256, 99,76% de l'optimal). La politique optimale est obtenue après la troisième itération : la valeur atteinte est de -1250. La quatrième itération est nécessaire, puisque la condition d'arrêt de l'algorithme est l'obtention de deux politiques successives identiques.

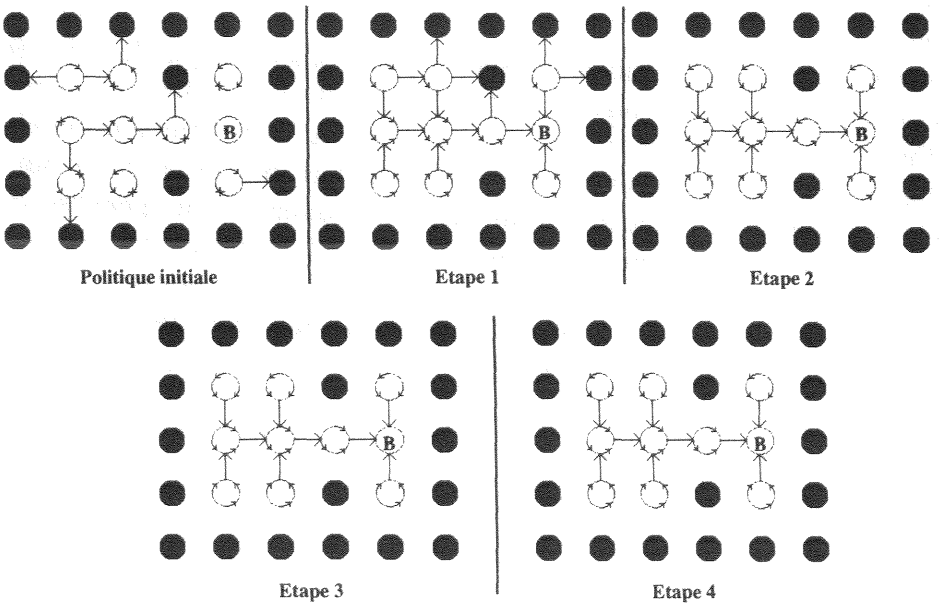


FIG. 3.8 – Policy Iteration : *d'une politique aléatoire à la politique optimale.*

Cette figure ne montre qu'un exemple de déroulement de l'algorithme. En effet, celui-ci dépend de la politique initiale, qui peut être plus ou moins bonne. Sur 10 exécutions de

l'algorithme, une a nécessité 3 itérations, deux ont convergé au bout de 5 itérations, et les sept restantes se sont achevées après 4 itérations. Le temps CPU moyen (sur un Sparc 10) a été de 0,35 secondes.

Value Iteration : politiques successives

L'algorithme *Value Iteration* a un fonctionnement très différent, plus « exploratoire » (voir le paragraphe 2.5.3, page 24). En effet, ici on cherche à approcher l'espérance de gain de chaque état sur un horizon infini. Au départ, tous les états ont une valeur nulle. Lors de la première itération, on calcule la valeur obtenue par l'exécution d'une seule action par état. La deuxième itération donne la valeur à l'horizon 2, c'est-à-dire si on exécute successivement deux actions, et ainsi de suite. La condition d'arrêt de l'algorithme est donc définie par un seuil de différence entre deux valeurs successives. Dans l'exemple donné ici, l'algorithme s'arrête quand la différence de valeur totale de deux politiques successives est inférieure à 10^{-4}.

La figure 3.9 montre les trois premières itérations de l'algorithme. La valeur de cette politique est de -36 : au temps $t = 0$, tous les états ont une valeur nulle, donc au temps $t = 1$, la valeur d'un état est définie ainsi : $V_1(s) = R(s)$. La valeur de chaque état à $t = 1$ est donc -1.

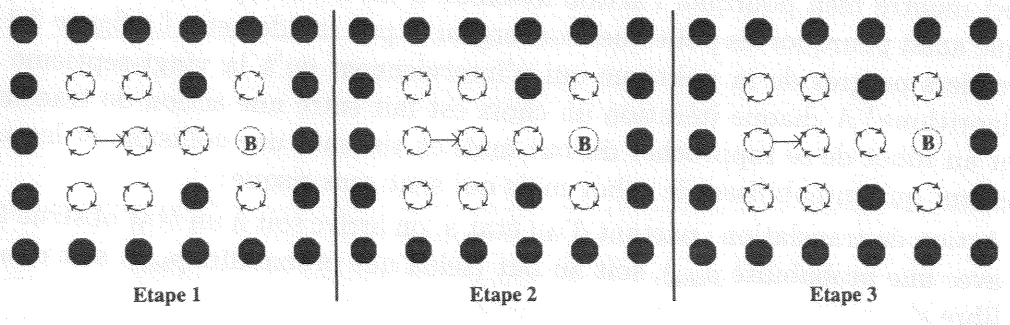


FIG. 3.9 – Value Iteration : *les trois itérations initiales.*

Deux remarques peuvent être faites sur cette figure : les trois politiques sont quasiment identiques, et seul un état a pour meilleure action `Avancer`. Ces deux faits sont d'ailleurs liés. S'il n'y a qu'une seule action `Avancer`, c'est parce qu'il n'y a qu'un seul état dans lequel cette action peut être prise sans faire chuter la valeur de l'état. En effet, si on examine la fonction de transition utilisée (voir la figure 3.1, page 36), on se rend compte que les actions de pivotement ne prévoient aucune transition hors de la position courante (seule l'orientation peut changer). L'exécution de ces actions est donc sans risque, contrairement à l'action de translation, qui peut amener à diverses situations de collision. La figure 3.10 (à gauche) montre les risques de collision qu'entraîne l'exécution d'une action de translation depuis l'état situé en colonne 1, ligne 1. Trois transitions amènent à une collision, avec une probabilité totale de 0,21. De ce fait, lors de la première itération le choix de l'action est fait ainsi :

- Action `Avancer` : $V_1(s) = R(s) + 0{,}21 \times V_{\texttt{OBS}} + 0{,}79 \times V_{\texttt{LIBRE}}$. Ici $V_{\texttt{OBS}}$ est la valeur d'un obstacle, c'est-à-dire -100 avec les paramètres que nous utilisons, et $V_{\texttt{LIBRE}}$ est

la valeur d'un état quelconque non obstrué au temps $t = 0$, c'est-à-dire 0. Cette action amène donc a une valeur de -22 pour cet état.

- Actions de pivotement : $V_1(s) = R(s) + 1 \times V_{\texttt{LIBRE}}$. La valeur est ici de -1, c'est donc une des deux actions de pivotement qui est choisie.

En revanche, dans l'état situé en dessous (colonne 1, ligne 2), l'exécution d'une action de translation n'amène que dans des états libres, comme on le voit sur la partie droite de la figure 3.10. De ce fait, quelle que soit l'action exécutée en cet état, sa valeur reste égale à -1. Si c'est l'action `Avancer` qui est choisie ici, ce n'est pas parce qu'elle est meilleure, mais tout simplement parce que c'est la première essayée par l'algorithme.

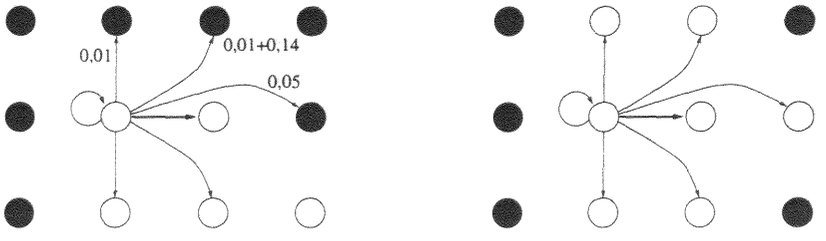


FIG. 3.10 – *Risque de collision depuis les états de coordonnées (1,1) et (2,1).*

Ceci montre bien pourquoi l'action `Avancer` n'est choisie qu'une seule fois, mais cela explique aussi pourquoi les politiques convergent si peu rapidement. La figure 3.11 montre les premiers progrès de la politique qui n'interviennent qu'à la vingt-septième itération de l'algorithme ! A chaque itération un choix est fait entre une action de translation, qui permet au robot de se rapprocher du but mais en risquant une collision, et les actions de pivotement qui immobilisent le robot mais qui sont sans risque :

- Action de translation : partant d'un état s, on arrive soit à un état obstrué (collision, avec une probabilité p_{obs}), soit au but (selon une probabilité p_{but}), soit dans un état libre s'.

$$V_{t+1}(s) = R(s) + \gamma \times [p_{obs} \times -100 + p_{but} \times 0 + (1 - p_{obs} - p_{but}) \times V_t(s')]$$

- Action de pivotement : d'un état s, on passe dans tous les cas à un état libre s'.

$$V_{t+1}(s) = R(s) + \gamma \times 1 \times V_t(s')$$

Lors des premières itérations, la valeur obtenue par l'exécution d'une action `Avancer` est très coûteuse par rapport à l'exécution d'une action de pivotement. En revanche, au fur et à mesure que le nombre d'itérations augmente, la valeur de l'état devient de plus en plus importante, et le coût de pivoter sur place devient plus important que le coût induit par l'exécution d'une action `Avancer`. A $t = n$, $n \geq 2$, l'exécution de n actions de pivotement donne une valeur $V_n(s) = -1 - \gamma - \gamma^2 - ... - \gamma^{n-1}$. Pour les états situés au-dessus et en dessous du but, c'est à $t = 27$ que cette valeur devient inférieure à la valeur obtenue par l'exécution d'une translation. De ce fait, l'action `Avancer` qui mène directement au but est enfin choisie. Puis, sur quelques itérations, la politique est encore légèrement améliorée, mais il faut attendre l'itération 42 pour arriver à une autre amélioration (figure 3.12), puis l'étape 50 (figure 3.13) et c'est enfin après la soixante-deuxième itération que la

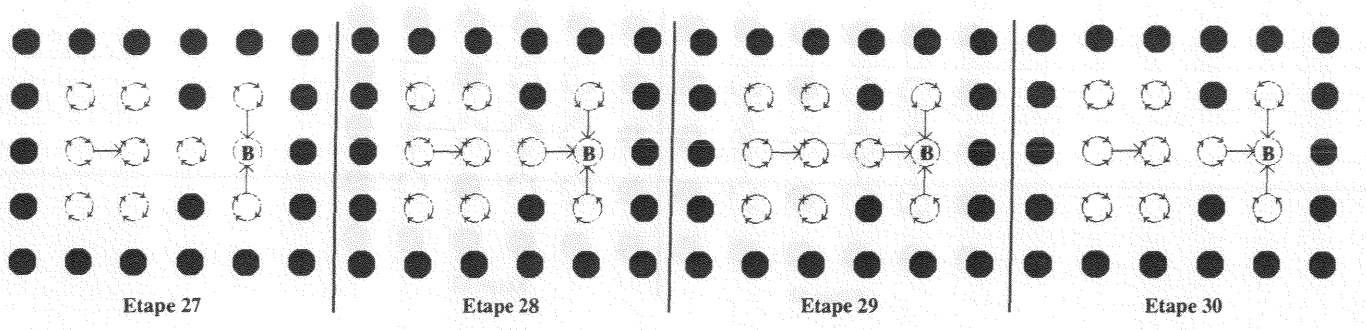


FIG. 3.11 – Value Iteration *: premières améliorations.*

politique optimale est trouvée (figure 3.14). A ce stade, la politique a une valeur de -1242, et ce n'est qu'à l'itération 89 que l'algorithme s'arrête avec une politique de valeur approximativement égale à -1250.

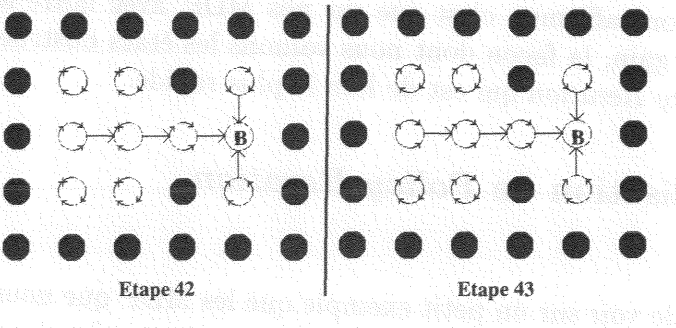


FIG. 3.12 – Value Iteration *: suite des améliorations.*

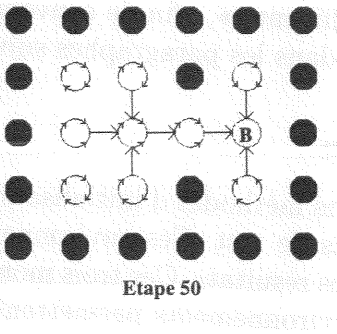


FIG. 3.13 – Value Iteration *: encore des améliorations.*

Cet exemple montre bien que *Value Iteration* est peu adapté à notre modèle : le nombre d'itérations nécessité se répercute sur le temps de calcul, puisque le temps d'exécution moyen (sur 10 exécutions) est de 4,1 secondes.

Conclusion

Ce petit exemple ne constitue en aucun cas une preuve de la supériorité de *Policy Iteration* vis-à-vis de *Value Iteration*. Par exemple, sur le même problème, en faisant

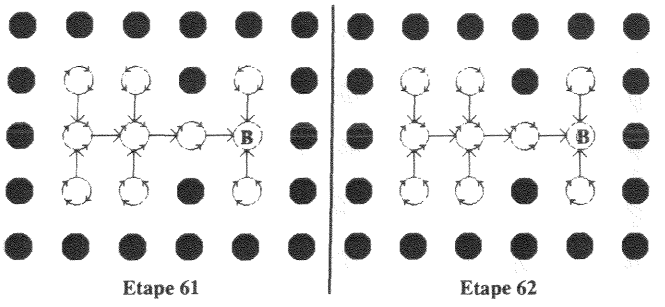


FIG. 3.14 – Value Iteration *: obtention de la solution optimale.*

uniquement passer le facteur γ de 0,99 à 0,9, l'écart entre les deux algorithmes a fortement diminué : le nombre d'itérations de *Value Iteration* est tombé à 36, pour un temps de calcul de 1,7 secondes, alors que les temps de calcul de *Policy Iteration* n'ont pas varié. Ceci montre bien l'importance des paramètres du MDP sur l'efficacité de ces algorithmes. Tout ce que nous pouvons affirmer, c'est que sur nos MDP, avec notre fonction de transition, notre fonction de gain, la façon dont nous valuons les états obstrués et un facteur γ fixé à 0,99, c'est *Policy Iteration* qui est de loin le plus rapide.

3.4.3 Initialisation de *Policy Iteration*

Intérêt

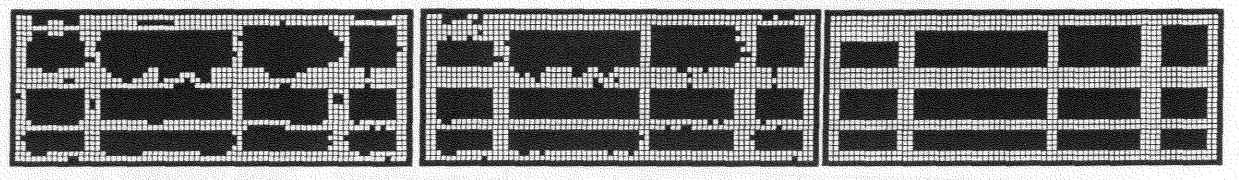
Nous venons de voir sur un petit exemple que les MDP que nous utilisons sont résolus plus rapidement par *Policy Iteration*, alors même que nous n'avons prêté aucune attention à la politique donnée en entrée de l'algorithme, qui est totalement aléatoire. Plutôt que de partir avec une telle politique, il serait sans doute intéressant d'initialiser l'algorithme avec une politique plus « intelligente », afin de converger plus rapidement vers l'optimal. C'est ce que nous examinons dans les paragraphes suivants.

Méthodologie d'évaluation

Afin de tester les différentes méthodes d'initialisation de politiques présentées ici, nous utilisons trois environnements de test. Ces environnements seront d'ailleurs repris dans chaque chapitre présentant des résultats. Ces trois mondes sont présentés figures 3.15, 3.16 et 3.17. Les deux premiers environnements permettent de tester les méthodes en fonction des obstacles contenus dans l'environnement. Dans ce but, ceux-ci ont été déclinés en trois versions : la première contient tous les obstacles, la dernière n'en contient aucun, alors que la seconde représente un compromis entre les deux. Dans chaque environnement, le but a été placé successivement en différentes positions : 20 pour le premier et 13 pour le second. Dans le troisième environnement, dont la structure est différente des deux premiers, le but a été déplacé 26 fois. Dans chaque cas, nous donnerons les chiffres suivants :

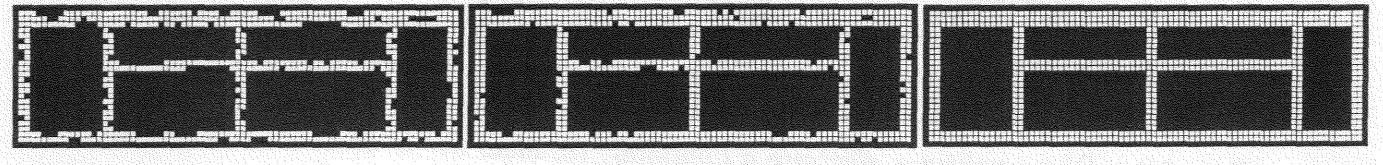
- Evaluation de la qualité de la politique initiale : comme vu au paragraphe 3.4.1, deux évaluations sont données (pourcentage de la valeur par rapport à la valeur de la politique optimale et pourcentage d'actions correctes) ;

- Temps de calcul de la solution optimale en partant de la politique initiale (temps CPU obtenus sur un quadri-processeur Sun Ultra Sparc) ;
- Nombre d'itérations de *Policy Iteration* nécessaires à l'obtention de la solution optimale.



a. 191 obstacles (3360 états) b. 81 obstacles (3800 états) c. Aucun obstacle (4124 états)

FIG. 3.15 – *Premier environnement de test.*



a. 130 obstacles (2192 états) b. 58 obstacles (2480 états) c. Aucun obstacle (2712 états)

FIG. 3.16 – *Deuxième environnement de test.*

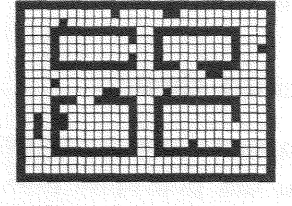


FIG. 3.17 – *Troisième environnement (1640 états).*

Par une politique aléatoire

Nous donnons ici les résultats obtenus en utilisant une politique initiale calculée de façon totalement aléatoire. Chaque problème a donné lieu à 10 exécutions, les résultats présentés sont donc une moyenne sur 200 exécutions pour le premier environnement, 130 sur le deuxième et 260 sur le troisième. Les résultats obtenus sont présentés tables 3.1, 3.2 et 3.3.

Quel que soit l'environnement, la politique aléatoire est très mauvaise. Même si une action sur trois est correcte, la valeur de la politique obtenue est très mauvaise. La qualité moyenne ne dépasse pas 3,9%, et la meilleure politique trouvée ne représente que 23,37%. Les temps de calcul sont plutôt impressionnants, et se dégradent au fur et à mesure que les obstacles sont enlevés : il faut en moyenne plus de 42 minutes de temps CPU pour passer de la solution aléatoire à la solution optimale sur le plus grand environnement. A ce stade, l'utilisation de politiques de ce type en planification semble illusoire. Heureusement, notre

	Tous les obstacles	Intermédiaire	Sans obstacle
Qualité moyenne	1,92	1,22	0,65
Qualité minimum	0,06	0,05	0,00
Qualité maximum	10,5	4,68	2,84
% Actions moyen	33,66	33,61	33,66
Temps de calcul	1463	1921	2526
Nombre d'itérations	12,63	14,03	15,73

TAB. 3.1 – *D'une politique aléatoire à l'optimal : résultats sur le 1er environnement.*

	Tous les obstacles	Intermédiaire	Sans obstacle
Qualité moyenne	3,90	2,91	1,55
Qualité minimum	0,00	0,00	0,00
Qualité maximum	23,37	13,56	6,46
% Actions moyen	33,56	33,71	33,66
Temps de calcul	691	818	1032
Nombre d'itérations	10,98	11,1	12,04

TAB. 3.2 – *D'une politique aléatoire à l'optimal : résultats sur le 2ème environnement.*

problème de planification peut être aisément initialisé de manière plus « intelligente », ce qui nous laisse espérer l'obtention plus rapide de solutions optimales. C'est ce que nous voyons aux paragraphes suivants.

Par un algorithme de plus court chemin

Vu les résultats obtenus en partant d'une politique initiale aléatoire, nous avons cherché à initialiser *Policy Iteration* par des politiques plus intéressantes. Comme nous l'avons vu au paragraphe 3.3.8 (page 38), les politiques calculées en utilisant nos paramètres préconisent des chemins qui peuvent être vus comme un compromis entre la distance à parcourir et la sécurité du robot sur les chemins. Notre première idée a donc été d'initialiser l'algorithme de résolution par une politique utilisant uniquement le critère du plus court chemin, que nous appelerons par la suite une politique « plus court chemin ». L'algorithme que nous utilisons est très classique. Les états adjacents au but se voient attribuer l'action menant au but avec la plus grande probabilité, puis les voisins de ces états adjacents sont traités, puis les voisins des voisins, et ainsi de suite jusqu'à ce que tout l'environnement soit traité. Cette procédure est détaillée par l'algorithme 4. La simplicité de cet algorithme permet d'obtenir une solution initiale très rapidement (moins d'un dixième de seconde de temps CPU), solution qui dans certains cas très particuliers peut correspondre exactement à la solution finale. C'est le cas par exemple du petit environnement de la figure 3.2 (page 37). De ce fait, une seule itération de *Policy Iteration* suffit pour obtenir la solution optimale. Dans le cas général, la politique *plus court chemin* est assez différente de la politique optimale, comme on le voit sur la figure 3.18 qui donne la politique *plus court chemin* calculée pour l'environnement de la figure 3.6 (page 39).

Néanmoins, le fait d'avoir une politique initiale de ce type permet de converger plus

Algorithme 4 Calcul d'une politique *plus court chemin* pour un MDP $\mathcal{M}$.

```
i : ENTIER
but_courant,voisin : ETAT
fbut : LISTE de ETAT
pi : TABLEAU de |S| ACTION
non_traite : TABLEAU de |S| BOOLEEN

//aucun état n'a encore été traité
pour tout s ∈ S faire
  non_traite[s] ⇐ VRAI
fin pour

f_AjouteFin(fbut,EtatBut(M)) //Ajout en fin de file de l'état but du MDP M
//Traitement de toute la file, tant qu'elle contient des états
tant que f_NonVide(fbut) faire
  but_courant ⇐ f_Defile(fbut) //le premier état de la file en est sorti

  //Traitement des 6 voisins de l'état courant à atteindre : 1 au-dessus, 1 au-dessous,
  //1 à gauche, 1 à droite, et 2 sur la même position mais orientés orthogonalement
  pour i = 1 à 6 faire
    voisin⇐ChercheVoisin(but_courant,i) //récupération du voisin numéro i
    //si ce voisin n'est pas un obstacle et qu'il n'a pas encore été traité, on s'en occupe
    si est_libre(voisin) ET non_traité[voisin] alors
      pi[voisin]⇐ arg max_a T(voisin,a,but_courant) //affectation de l'action menant
      au but courant
      non_traite[voisin]⇐FAUX //ce voisin a été traité, on le marque
      f_AjouteFin(fbut,voisin) //ajout en fin de file du voisin s'il n'y est pas encore
    fin si
  fin pour
fin tant que
```

Qualité moyenne	1,24
Qualité minimum	0,04
Qualité maximum	7,41
% Actions moyen	33,64
Temps de calcul	195
Nombre d'itérations	10,95

TAB. 3.3 – *D'une politique aléatoire à l'optimal : résultats sur le 3ème environnement.*

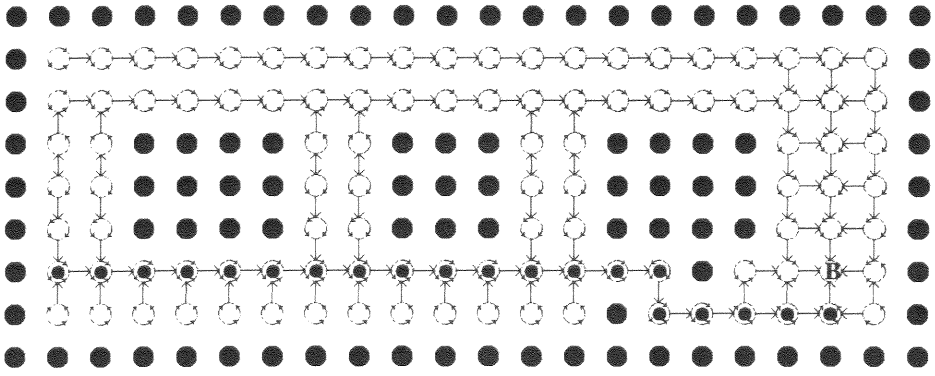


FIG. 3.18 – *Exemple de politique* plus court chemin.

rapidement vers la solution optimale, comme on le voit dans les tables 3.4, 3.5 et 3.6. La qualité des politiques calculées est bien meilleure : en moyenne plus de 92% par rapport à l'optimal pour la première instance du deuxième environnement, alors que le plus mauvais résultat obtenu reste supérieur à 26%. La figure 3.19 compare les politiques aléatoires et *plus court chemin* sur nos trois environnements de test (à gauche les trois instances de la figure 3.15, puis les trois instances de la figure 3.16, puis la figure 3.17). Dans tous les cas, la politique *plus court chemin* est beaucoup plus intéressante, et ce résultat se répercute sur les temps de calcul de la solution optimale, comme le montre la figure 3.20. En moyenne, le temps nécessaire à la convergence de l'algorithme est réduit d'un quart, et sur nos sept exemples le gain varie entre 32,5% et 16%. Les environnements sur lesquels le gain de temps est le plus important sont ceux comportant beaucoup d'obstacles, puisque c'est pour ceux-là que la politique *plus court chemin* est la meilleure. En effet, on observe une dégradation des résultats dans les environnements peu obstrués, comme on le constate dans les tables 3.4 et 3.5. Dans l'instance la plus obstruée du premier environnement, la qualité moyenne est 79,29%, elle passe à 63,8% pour la seconde instance et chute à 44,8% pour la dernière. Le second environnement montre le même comportement : on passe de 92,75% à 85,56% puis à 63,60%. C'est aussi à chaque fois dans l'instance ne comportant aucun obstacle que la politique la moins bonne a été calculée. Ces faits s'expliquent aisément puisque nos politiques utilisent comme critère d'optimalité unique la distance à parcourir jusqu'au but. Or, comme nous l'avons vu au paragraphe 3.3.8, les politiques optimales calculées avec nos paramètres génèrent des chemins situés au milieu des couloirs. Dans un environnement obstrué, les chemins sont peu nombreux ; la politique *plus court chemin* évite les obstacles, et ressemble beaucoup à la politique optimale. En revanche,

dans les environnements comportant peu d'obstacles, la politique *plus court chemin* génère des chemins sur lesquels le robot longera les murs : aucun effort n'est fait pour aller se placer dans une zone de sécurité, comme le fait la politique optimale (on le voit bien dans le couloir vertical de droite des figures 3.6 et 3.18). De ce fait, les politiques *plus court chemin* sont dans ce cas de mauvaise qualité. Pour tenter de corriger ce défaut, nous introduisons un second algorithme chargé de calculer les solutions initiales, algorithme que nous présentons au paragraphe suivant.

	Tous les obstacles	Intermédiaire	Sans obstacle
Qualité moyenne	79,29	63,80	44,80
Qualité minimum	65,29	48,41	26,78
Qualité maximum	88,87	80,41	61,16
% Actions moyen	70,01	66,36	59,37
Temps de calcul	1069	1616	2010
Nombre d'itérations	8,15	10,35	10,95

TAB. 3.4 – *D'une politique* plus court chemin *à l'optimal : résultats sur le 1er environnement.*

	Tous les obstacles	Intermédiaire	Sans obstacle
Qualité moyenne	92,75	85,56	63,60
Qualité minimum	82,75	70,12	34,00
Qualité maximum	97,60	96,41	86,53
% Actions moyen	83,43	73,67	63,38
Temps de calcul	466	615	862
Nombre d'itérations	6,54	7,38	8,77

TAB. 3.5 – *D'une politique* plus court chemin *à l'optimal : résultats sur le 2ème environnement.*

Qualité moyenne	75,83
Qualité minimum	51,21
Qualité maximum	86,89
% Actions moyen	73,84
Temps de calcul	134
Nombre d'itérations	7,38

TAB. 3.6 – *D'une politique* plus court chemin *à l'optimal : résultats sur le 3ème environnement.*

Ajout de la prise en compte des obstacles

L'algorithme précédent présentant le défaut de ne pas préconiser des chemins sûrs, éloignés des murs, nous l'avons légèrement modifié pour prendre en compte ces comportements. Dans la suite de ce mémoire, nous désignerons les politiques calculées par cet

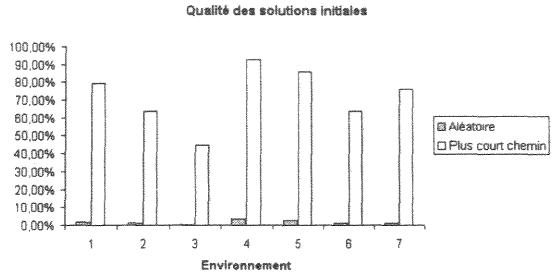


FIG. 3.19 – *Comparaison des qualités des politiques aléatoires et* plus court chemin.

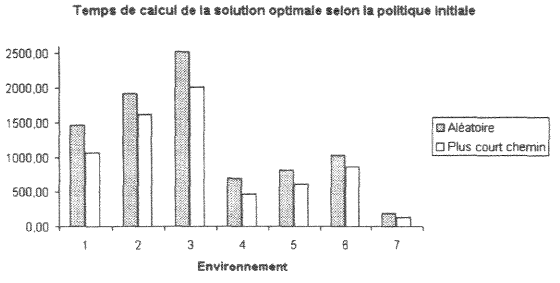


FIG. 3.20 – *Comparaison des temps de calcul selon la politique initiale.*

algorithme sous le nom de politiques *plus court chemin adapté*. Afin de conserver la rapidité de calcul de la politique initiale, nous avons conservé quasiment le même algorithme, à un détail près : avant d'ajouter les voisins du but courant à la file des états à traiter, nous les trions selon le nombre d'obstacles auxquels ils sont adjacents. Par exemple, lors de la première itération de la boucle, les voisins du but global de l'environnement sont ajoutés dans la file dans l'ordre suivant : voisin du dessus (aucun obstacle adjacent), voisin de gauche (un obstacle de biais), puis voisins de droite et du dessous (un obstacle adjacent). Ce simple tri permet de générer les comportements d'éloignement des murs, comme on le voit dans le couloir de droite de la figure 3.21. Cette politique représente 94,47% de la politique optimale, alors que l'algorithme précédent donnait une valeur de 84,54%. Cette politique se rapproche de la politique optimale, même si les chemins n'évitent pas encore les couloirs obstrués comme a tendance à le faire la politique optimale (figure 3.6, page 39).

L'apport constaté sur ce petit environnement est également constaté sur nos environnements de test, comme on peut le voir sur la figure 3.22. Ici aussi, les trois instances du premier environnement sont à gauche, puis viennent les trois instances du deuxième, puis le troisième. En moyenne, la qualité est passée de 72,23% à 80,32%. Les résultats brut sont présentés dans les tables 3.7, 3.8 et 3.9. C'est surtout sur les environnements peu obstrués que la qualité augmente (pour l'instance la moins obstruée, on passe de 44,8% à 56,04% sur le premier environnement et de 63,6% à 76% sur le second). En comparaison, les environnements comportant beaucoup d'obstacles, qui donnaient déjà de bons résultats avec une politique *plus court chemin*, bénéficient d'une amélioration moindre.

En revanche, ici la hausse de qualité assez sensible des politiques générées ne s'ac-

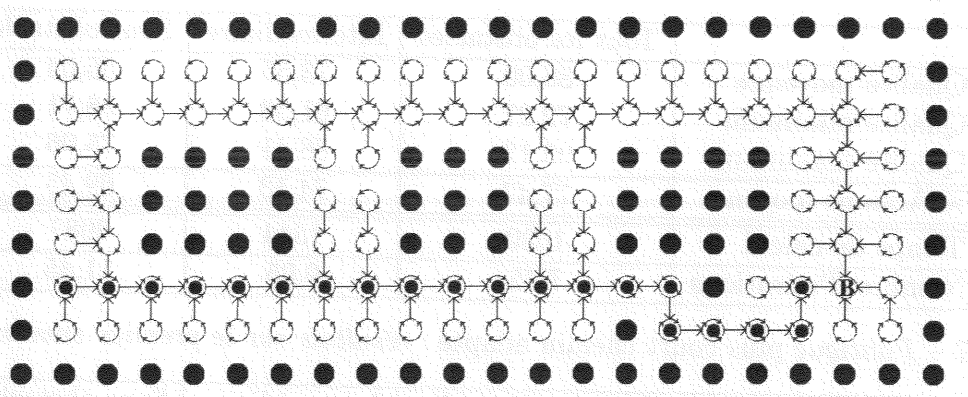


FIG. 3.21 – *Placement au milieu des couloirs.*

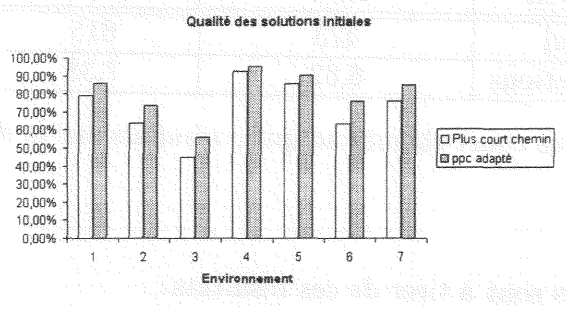


FIG. 3.22 – *Comparaison des qualités des politiques* plus court chemin *et* plus court chemin adapté.

compagne pas d'un gain en temps de calcul de la solution optimale très impressionnant (figure 3.23). En effet, le temps de calcul moyen sur nos sept environnements de test passe de 968 à 960 secondes. Sur deux environnements, le temps de calcul est même supérieur, alors que les politiques initiales sont de meilleure qualité. Ceci n'est pas dû à un caprice de l'horloge : c'est en fait le nombre d'itérations nécessaire à la convergence qui augmente. Par exemple, sur la troisième instance du premier environnement, il faut en moyenne 10,95 itérations (ce qui donne un temps de calcul de 2010 secondes) en partant de politiques *plus court chemin* (de qualité moyenne de 44,8%), alors qu'en partant de politiques calculées grâce à ce second algorithme (qualité 56,04%), il a fallu en moyenne 11,45 itérations (2087 secondes). Il semble en fait que le nombre d'itérations nécessaire à la convergence de l'algorithme ne dépende pas uniquement de la qualité de la politique initiale. Selon la « structure » de la politique initiale, les politiques sont peut-être plus faciles à améliorer, mais nous ne pouvons pas expliquer ce phénomène. Malgré tout, le bilan de cet algorithme n'est pas mauvais : les politiques obtenues sont meilleures que celles générées par l'algorithme précédent, alors que le temps de calcul est sensiblement le même.

	Tous les obstacles	Intermédiaire	Sans obstacle
Qualité moyenne	86,00	73,55	56,04
Qualité minimum	72,27	61,33	38,71
Qualité maximum	93,12	86,41	75,96
% Actions moyen	77,47	74,13	69,45
Temps de calcul	1020	1544	2087
Nombre d'itérations	7,75	9,95	11,45

TAB. 3.7 – *Politique* plus court chemin adapté *: résultats sur le premier environnement.*

	Tous les obstacles	Intermédiaire	Sans obstacle
Qualité moyenne	95,21	90,05	76,00
Qualité minimum	86,49	76,33	44,63
Qualité maximum	99,47	97,25	97,14
% Actions moyen	85,41	80,12	72,08
Temps de calcul	473	615	845
Nombre d'itérations	6,62	7,38	8,62

TAB. 3.8 – *Politique* plus court chemin adapté *: résultats sur le deuxième environnement.*

Conclusion

Deux enseignements sont à tirer de ces résultats :

- Choisir comme politique initiale une politique π_1 de valeur V_{π_1} plutôt qu'une politique π_2 de valeur V_{π_2} avec $V_{\pi_2} < V_{\pi_1}$ ne garantit pas l'obtention plus rapide de la politique optimale ;
- Si le passage d'une politique initiale très mauvaise (aléatoire) à une politique initiale de « bonne » qualité (*plus court chemin*) a permis de réduire le temps de calcul de 25%, le passage à une politique encore meilleure n'a lui pratiquement pas fait évoluer le temps de calcul.

Ceci est dû au fonctionnement de *Policy Iteration*. Lors des premières itérations, la qualité des politiques augmente énormément. En revanche, les dernières itérations sont passées à régler des détails, qui font très peu augmenter la qualité des politiques. S'il est intéressant de prendre comme politique initiale une politique représentant 75% de l'optimale plutôt

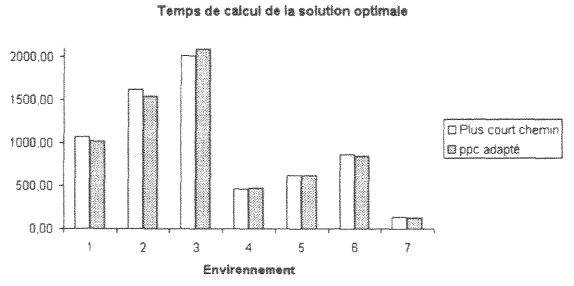


FIG. 3.23 – *Comparaison des temps de calcul selon la politique initiale.*

Qualité moyenne	85,41
Qualité minimum	72,12
Qualité maximum	94,11
% Actions moyen	82,10
Temps de calcul	131
Nombre d'itérations	7,23

TAB. 3.9 – *Politique* plus court chemin adapté *: résultats sur le troisième environnement.*

qu'une politique représentant 5%, en revanche la différence entre deux politiques représentant respectivement 75% et 85% de l'optimal est pratiquement nulle en ce qui concerne le temps de convergence vers la solution optimale.

Malgré tout, le calcul rapide de solutions sous-optimales n'a pas l'unique intérêt de permettre un calcul rapide de la solution optimale. Une « bonne » solution sous-optimale peut être exécutée à la place de la politique optimale, pendant que la politique optimale est calculée en parallèle, comme c'est fait par exemple dans [Dean *et al.*, 1993]. Plus la politique sous-optimale sera proche de l'optimale, plus sûre devrait être l'exécution. Dans ce cadre, la recherche d'autres méthodes de calcul de bonnes solutions sous-optimales est une piste intéressante. La suite de ce mémoire sera principalement consacrée à cela.

3.5 Exécution d'une politique

Dans les paragraphes précédents, nous avons exposé les choix que nous avons fait en ce qui concerne le modèle et l'algorithme de planification, mais nous n'avons pas encore évoqué l'exécution des politiques calculées. Nous avons choisi d'utiliser un MDP parfaitement observé, cadre dans lequel on considère que l'état courant de l'environnement est connu. Ceci n'est évidemment pas vrai pour un robot mobile évoluant dans un laboratoire, c'est-à-dire dans des bureaux et couloirs peu différentiables les uns des autres. La bonne exécution des politiques ne peut être envisagée sans l'ajout de différents modules, chargés notamment de la localisation et de l'évitement d'obstacles imprévus, pour lesquels la politique ne prévoit aucune stratégie d'évitement. C'est à ces modules d'exécution que nous nous intéressons dans ce paragraphe.

3.5.1 Utilisation d'un navigateur

Lorsqu'on exécute une politique, on ne peut pas oublier le caractère dynamique de l'environnement et espérer que tout fonctionnera comme prévu. Le robot peut être confronté à des obstacles non prévus lors de la phase de construction de la politique, voire à des obstacles mobiles. Il arrive également, du fait du caractère bruité des capteurs, que le robot soit mal localisé. L'exécution de l'action optimale correspondant à l'état courant peut alors devenir dangereuse si cet état n'est pas l'état réel du robot. Toutes ces raisons expliquent le besoin d'un module de navigation pour piloter notre robot. Nous avons eu la chance de pouvoir utiliser le navigateur écrit dans l'équipe MAIA par Olivier Aycard [Aycard *et al.*, 1997b]. Celui-ci est très bien détaillé dans [Aycard, 1998], au chapitre 6

(les figures de ce paragraphe sont d'ailleurs tirées de ce document). Pour résumer, celui-ci a pour mission de déplacer réactivement le robot vers un but, en fonction des obstacles rencontrés sur son chemin. Le comportement global du robot est obtenu en intégrant les comportements locaux, chaque comportement local étant calculé en fonction d'un ensemble différent de capteurs. Les capteurs utilisés sont les infra-rouges et les ultra-sons, dont les données sont fusionnées. Ces capteurs sont divisés en cinq zones (figure 3.24) : une zone frontale comportant un capteur, une zone avant gauche et une zone avant droit comportant chacune deux capteurs et une zone de chaque côté du robot, chaque zone comportant trois capteurs. L'utilisation de techniques de logique floue permet d'associer aux données renvoyées par les capteurs une valeur abstraite, dont le domaine est : {Très Proche, Proche, Loin}. Pour chaque zone un comportement local est défini à l'aide de règles. Par exemple, pour le côté gauche, on utilise trois règles :

```
Si côté gauche est très proche alors tourner rapidement à droite
Si côté gauche est proche alors tourner lentement à droite
Si côté gauche est loin alors ne pas tourner
```

Chaque comportement local va donc préconiser une rotation pour la base et la tourelle du robot. Lors de l'intégration des comportements locaux, une moyenne est faite sur ces degrés de rotation, et c'est cette rotation moyenne qui va être appliquée au robot. La vitesse de translation du robot est calculée en fonction des vitesses de rotation : plus le robot tourne, moins vite il avance. Cela s'explique par le fait que les rotations sont utilisées pour éviter les obstacles, il s'agit donc d'être prudent.

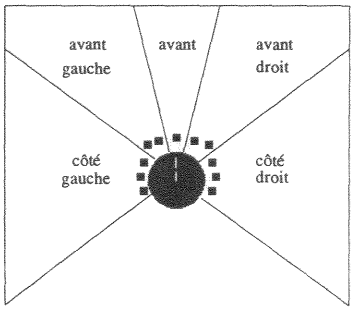


FIG. 3.24 – *Les cinq zones physiques correspondant aux ensembles de capteurs.*

Un exemple de comportement du navigateur est fourni par la figure 3.25 : le robot se déplace dans un couloir comportant des obstacles et une étagère. Les obstacles sont bien évités et le robot se place naturellement dans une zone de sécurité au milieu du couloir, afin d'éviter les risques de collision. Nous verrons en fin du chapitre suivant que ce comportement peut être très intéressant dans le cadre de l'agrégation d'états.

3.5.2 Localisation du robot

Une politique donnée par la résolution d'un MDP est un ensemble de couples (état,action), spécifiant dans chaque état une action optimale à exécuter. Lors de l'exécution d'une politique, la position courante du robot est très rarement connue avec certitude, mais on a

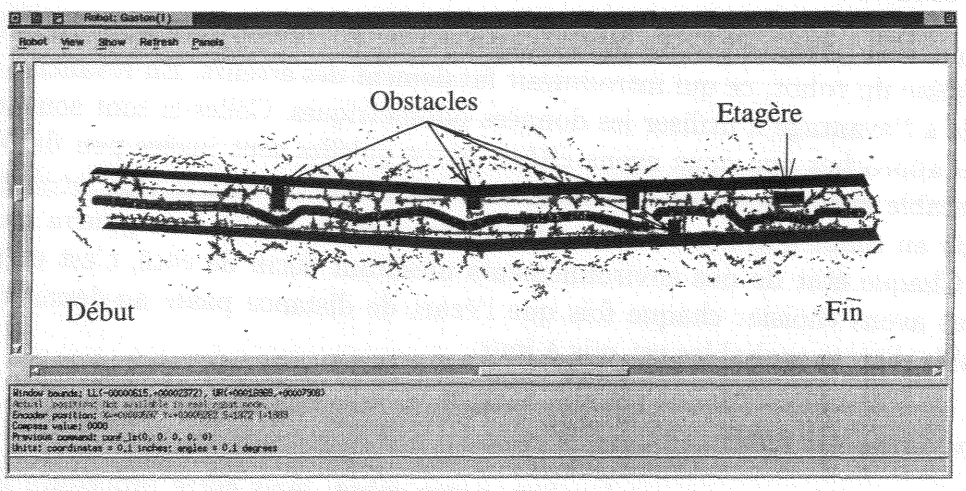


FIG. 3.25 – *Traversée d'un couloir.*

généralement une idée approximative de la localisation du robot. Pour résoudre ce problème, on utilise généralement un ensemble d'états probables (on parle alors de *belief state* ou *Markov localization*) qui contient les états dans lesquels le robot peut éventuellement se trouver. On se trouve donc à la frontière des POMDP, puisque cet ensemble est le même que celui utilisé pour représenter l'espace d'états dans ce cadre. Cet ensemble est mis à jour en fonction des positions probables précédentes, des données fournies par les capteurs et de la fonction de transition. La probabilité $l(s')$ de se trouver dans l'état s' alors qu'on a exécuté l'action a puis observé l'observation abstraite o est calculée de la façon suivante :

$$l(s') = \frac{O(s',o) \sum_{s \in \mathcal{S}} l_{\mathtt{prec}}(s) T(s,a,s')}{\alpha}$$

Ici $l_{\mathtt{prec}}$ est l'ensemble d'états probables calculé à l'étape précédente, et α est un facteur de normalisation. On retrouve également la fonction O classiquement utilisée lorsqu'on manipule un MDP partiellement observé. Cet ensemble d'états va permettre de choisir l'action à exécuter. Pour cela, diverses stratégies sont possibles [Cassandra *et al.*, 1996] : exécution de l'action correspondant à l'état ayant la plus forte probabilité, exécution de l'action la plus souvent optimale pour les états probables, etc. Dans les travaux étudiés, c'est quasiment toujours la première solution qui est choisie, nous avons donc fait de même.

Mise à jour de l'ensemble d'états probables

Ce qui n'est pas dit ici c'est quand est mis à jour cet ensemble. En théorie, il est recalculé après terminaison de l'action à exécuter. Il faudrait donc exécuter une action, arrêter le robot, calculer l'ensemble d'états probables, puis faire repartir le robot, et ainsi de suite. Un robot devant traverser un couloir de dix mètres de long aurait un bien étrange comportement, sans parler des problèmes que cela pourrait occasionner aux moteurs du robot. Nous avons donc décidé d'arrêter uniquement le robot lorsqu'il a une action de pivotement à effectuer ; les actions de translation sont faites de manière continue. On a donc le choix entre mettre à jour l'ensemble d'états probables à intervalles de temps réguliers

ou chaque fois qu'une distance d a été parcourue. La première solution est dépendante de la vitesse du robot, ce qui introduirait fatalement des erreurs. En revanche, la seconde méthode a l'avantage d'utiliser les données odométriques. Celles-ci sont souvent ignorées dans les approches que nous avons citées, parce qu'elles sont jugées peu fiables. Si c'est incontestable sur de longues distances, cela est beaucoup moins vrai localement : si on demande au robot d'avancer de 50 centimètres, l'écart constaté ne pourra pas être bien grand. Chaque état de nos environnements mesurant 50cm de côté, c'est cette distance que nous avons choisie : chaque fois que l'écart de distance passe au-dessus de ce seuil, l'ensemble d'états probables est mis à jour.

Observations du robot

Nous savons donc comment localiser notre robot, mais cette procédure fait appel à une fonction d'observation, que nous n'avons pas encore définie. Celle-ci affecte à chaque couple (état,observation) une probabilité représentant la probabilité que les capteurs du robot fassent telle observation dans tel état. La première chose à faire est donc de définir les observations possibles dans nos environnements. On peut pour cela utiliser des balises artificielles [Koenig et Simmons, 1996a] ou naturelles [Aycard *et al.*, 1997a] : couloir, intersection, porte ouverte ou fermée, etc. Le problème des balises naturelles est qu'elles sont difficilement modélisables et qu'il n'est pas toujours évident de les distinguer les unes des autres (par exemple une intersection par rapport à une porte ouverte), comme nous avons pu nous en apercevoir lors de notre collaboration avec Olivier Aycard, ancien doctorant de l'équipe MAIA (voir le paragraphe 3.6). Nous avons donc choisi d'utiliser des observations abstraites de bas-niveau, en utilisant les distances des capteurs. Le navigateur utilisé pour le pilotage de l'exécution (voir le paragraphe précédent) représente les perceptions du robot à l'aide de valeurs abstraites {Très Proche,Proche,Loin} et divise la ceinture de capteurs en cinq zones. De ces cinq zones, nous en avons gardé trois : avant du robot (cinq capteurs, combinaison des zones Avant, Avant gauche et Avant droit), côté gauche (trois capteurs) et côté droit (trois capteurs). Ceci nous a permis d'obtenir un ensemble de 3^3 observations abstraites, chacune étant plus ou moins probable dans les états de l'environnement.

Une fois cet ensemble d'observations défini, il nous faut définir la fonction d'observation, qui permet de connaître la probabilité respective d'une observation à partir d'un état. L'idéal serait d'apprendre cette fonction comme cela a été fait pour la fonction de transition, mais nous n'avons pas eu l'opportunité de le faire. Au lieu de cela, nous calculons ces probabilités automatiquement, en utilisant quelques données expérimentales. Tout d'abord, une lecture de la carte de l'environnement nous permet de définir l'observation « réelle » que devrait faire le robot à partir de chaque état. Cette observation réelle nous permet ensuite de calculer la probabilité de toutes les autres observations, en utilisant quelques règles simples :

Si observation réelle est `Loin` Alors
  Probabilité d'observer `Loin` : 0,8
  Probabilité d'observer `Proche` : 0,18
  Probabilité d'observer `Très proche` : 0,02

Si observation réelle est `Proche` Alors
Probabilité d'observer `Loin` : 0,12
Probabilité d'observer `Proche` : 0,7
Probabilité d'observer `Très proche` : 0,18

Si observation réelle est `Très Proche` Alors
Probabilité d'observer `Loin` : 0,02
Probabilité d'observer `Proche` : 0,18
Probabilité d'observer `Très proche` : 0,8

La combinaison de ces règles permet de calculer la probabilité de faire une observation dans un état. Par exemple, à partir d'un état situé au milieu d'un hall, dont l'observation réelle est constituée par le triplet {`Loin, Loin, Loin`}, la probabilité d'observer {`Très Proche, Proche, Très Proche`} est 0,02×0,18×0,02. Ce mode de fonctionnement simple permet de passer le moins de temps possible à calculer l'ensemble d'états probables. En effet, pour effectuer ce calcul, on retire la main au navigateur, laissant ainsi le robot sans surveillance. Nous avons donc volontairement simplifié la phase d'observation de l'environnement, surtout au vu des méthodes de plus haut-niveau que nous avons eu l'occasion d'étudier (voir le paragraphe 3.6). De ce fait, la localisation n'est pas toujours parfaite, on le verra ultérieurement.

3.5.3 Architecture utilisée

Le navigateur ayant été écrit en C++, comme tous nos programmes, il a pu aisément être réutilisé. Puisque nous arrêtons le robot avant chaque action de pivotement, le navigateur a uniquement été utilisé lorsque le robot est en train d'avancer. La figure 3.26 résume le fonctionnement de notre approche. On commence par déterminer l'action à exécuter, donnée par la politique optimale π en fonction de l'état courant s. Si celle-ci est une translation, c'est le navigateur qui s'en occupe. Celui-ci se charge de recueillir les données des capteurs du robot, et lorsque plus de 50cm ont été parcourus (Δ_d sur la figure), la main est passée à l'estimateur d'état qui recalcule l'ensemble d'états probables, ce qui permet de déterminer la prochaine action à exécuter. On boucle ainsi jusqu'à ce que le but soit atteint, ou qu'une collision ait interrompu la mission. Quand l'action à exécuter est un pivotement à 90°, le robot est arrêté, l'action exécutée et les nouvelles données des capteurs recueillies, puis on recalcule l'ensemble d'états probables, sans passer par le module de navigation.

3.5.4 Exemples d'exécutions

L'utilisation de techniques stochastiques en robotique est intéressante à plusieurs niveaux, que nous avons déjà évoqués : d'une part, les chemins optimaux sont calculés en évitant les passages obstrués, et d'autre part le pilotage utilisant les fonctions de transition et d'observation est une aide à la localisation. Même quand le robot se perd, il arrive

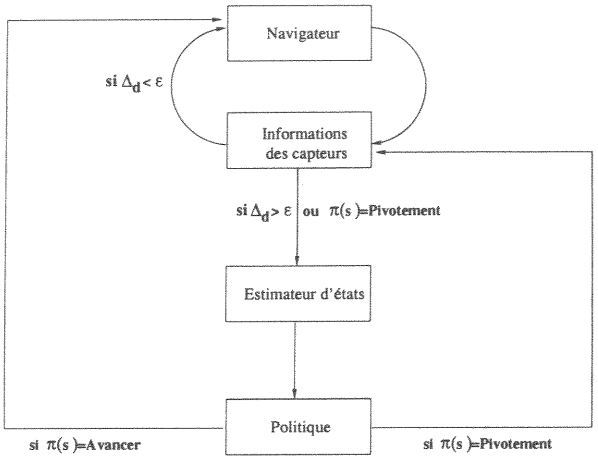


FIG. 3.26 – *Notre architecture d'exécution de politique.*

relativement souvent qu'après avoir exécuté plusieurs actions, il rencontre une observation discriminante dans l'environnement, ce qui lui permet de se relocaliser. La figure 3.27 montre deux traces d'exécution de la politique optimale présentée figure 3.6 (page 39). On voit sur la figure de droite que le robot, arrivé en début du couloir menant au but, a été perturbé par l'obstacle. Celui-ci n'obstruant que partiellement l'état, l'observation attendue n'a pas été faite, introduisant ainsi une erreur de localisation. Le robot est alors parti en sens inverse, avant d'atteindre l'intersection précédente, qui a été suffisamment discriminante pour que le robot se recale. Nous avons effectué dix exécutions de cette politique, et les dix missions ont réussi.

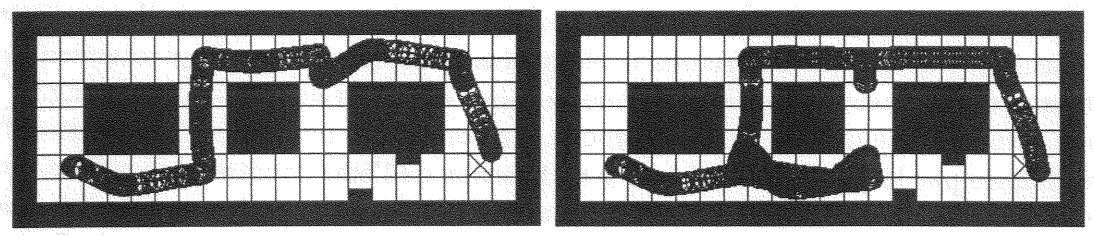


FIG. 3.27 – *Deux exécutions de la même politique optimale.*

La figure 3.28 montre la trace d'exécution d'une politique *plus court chemin* (figure 3.18, page 50) pour le même environnement. Dans le premier cas, le robot réussit à négocier la chicane et à atteindre son but. En revanche dans le second cas, il tarde à se réorienter vers le milieu du couloir, entraînant ainsi une collision avec le mur. Sur dix exécutions de cette politique, seules six ont réussi, et ont été à peine plus rapides que les exécutions de la politique optimale.

3.6 Localisation à l'aide de balises naturelles

En parallèle de notre travail centré sur les Processus Décisionnels de Markov, nous avons été amenés à collaborer au travail d'Olivier Aycard, alors doctorant dans l'équipe

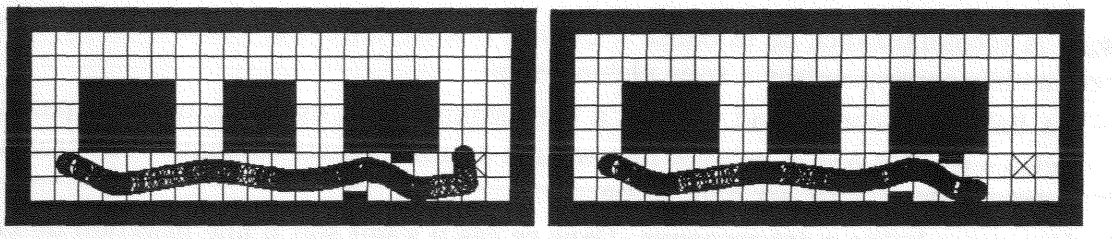


FIG. 3.28 – *Deux exécutions de la même politique* plus court chemin.

MAIA. Ce travail ayant fait l'objet d'une publication commune [Aycard *et al.*, 1998], nous le résumons brièvement ici. Le lecteur intéressé pourra se référer à [Aycard, 1998], qui présente ces recherches plus en détail (deuxième partie du mémoire). La base de ces travaux est l'utilisation de Modèles de Markov Cachés du Second Ordre (MMC2) [Mari, 1996; Mari *et al.*, 1997] pour l'apprentissage et la reconnaissance de balises naturelles de l'environnement (couloir, intersections de différents types, portes ouvertes ou fermées, etc.) Une fois les algorithmes d'apprentissage et de reconnaissance implantés et testés, j'ai pu contribuer aux travaux d'Olivier pour toute la partie localisation. L'environnement est ici modélisé par une carte topologique, le découpage en états étant effectué selon la balise naturelle à observer. Par exemple, si le robot est placé dans le couloir de gauche (numéro 1) de l'environnement de la figure 3.29, avec pour mission d'aller dans le couloir numéro 5, ce problème sera représenté par le modèle de la figure 3.30.

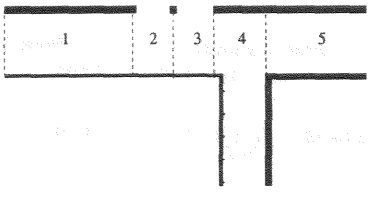


FIG. 3.29 – *Une partie d'un environnement.*

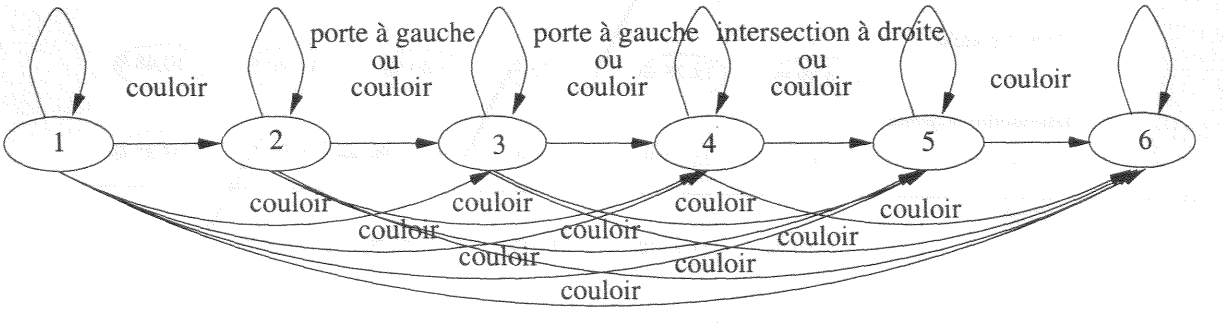


FIG. 3.30 – *Représentation à l'aide d'un MMC.*

Sur ce modèle, l'état numéro 6 représente la mission remplie. La technique de reconnaissance utilisée ne permet de reconnaître que le robot est dans un couloir qu'à l'instant où celui-ci a été quitté. Ainsi, la détection du passage de l'état 1 à l'état 2 ne se fera que

lorsque la reconnaissance de la balise « Couloir » a été faite. Les transitions entre états sur ce modèle correspondent à trois phénomènes :

- La balise qui correspond à l'état a été reconnue. Par exemple, on peut passer de l'état 4 à l'état 5 si la balise « Intersection à droite » a été reconnue. Une transition entre ces deux états est donc créée.
- Une autre balise a été reconnue. Dans ce cas il s'agit soit d'une insertion de balise (on reconnaît quelque chose qui n'existe pas) soit d'une mauvaise interprétation de balise. Dans ce cas, le modèle reste dans le même état (auto-transition).
- Une balise est invisible. En effet, une intersection peut être obstruée ou une porte fermée. Dans ce cas, le module de reconnaissance ne pourra pas la signaler, mais signalera au contraire qu'il a vu un couloir. De ce fait, il faut ajouter une transition de chaque état vers tous les états suivants lorsqu'on observe un couloir.

Une fois ce modèle construit, il ne nous reste plus qu'à utiliser la matrice de confiance obtenue par apprentissage pour piloter l'exécution. Cette matrice spécifie la probabilité d'être réellement passé devant une balise sachant la balise reconnue par le robot. Ceci permet de déterminer l'état courant du robot en fonction des observations faites. Par exemple, si le robot a reconnu durant sa mission les balises {Porte à droite, Couloir, Porte à gauche, Intersection à droite, Porte à gauche, Couloir}, on peut en déduire les états par lesquels il est successivement passé en utilisant l'algorithme *forward-backward*. La localisation du robot lors de la reconnaissance des balises successives est donnée par la figure 3.31

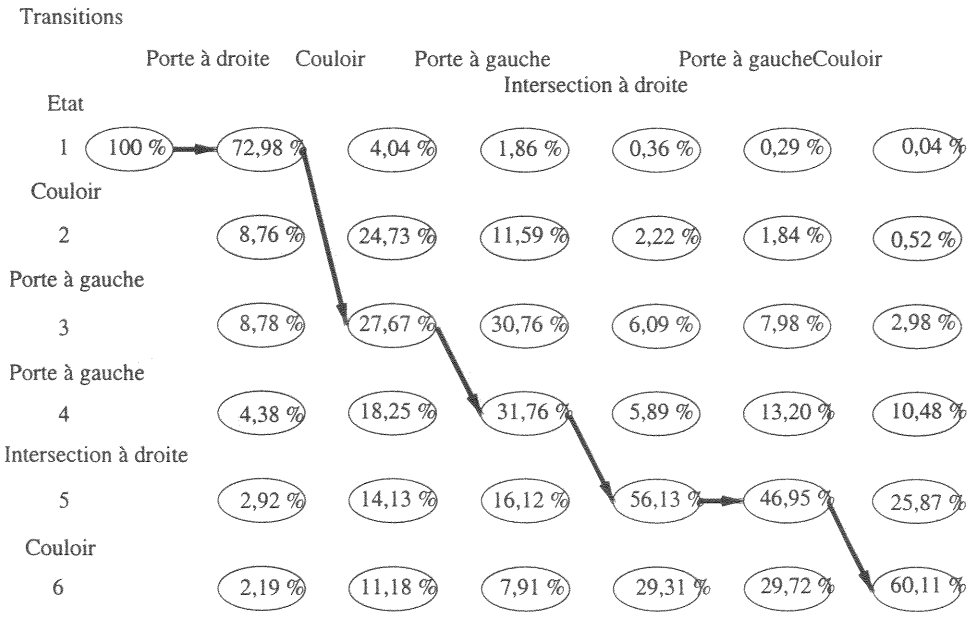


FIG. 3.31 – *Localisation en fonction des balises reconnues.*

Sur cette figure, seules les transitions les plus probables ont été représentées. Au départ, le robot est placé dans l'état 1, avec une confiance de 100%. La première balise reconnue est une porte à droite. Vu la configuration de l'environnement, il s'agit sans doute d'une insertion de balise, le robot est en fait toujours dans le même état (avec une probabilité

de 72,98%). La seconde balise observée est un couloir, ce qui indique que le robot est sorti de l'état 1. On est alors dans un des états suivants, et c'est l'état 3 qui a la plus forte probabilité. L'algorithme est ainsi exécuté jusqu'à arriver au but. En fonction des balises reconnues, le robot est très probablement passé par les états (1,1,3,4,5,5,6).

Cette étude nous a permis d'appréhender la complexité de la reconnaissance des balises naturelles, et nous a conforté dans l'idée d'utiliser un modèle de perception très simple. En effet, si cette approche est fiable, elle a le défaut de ne reconnaître les balises qu'une fois que le robot est passé devant. Par exemple, si le robot doit tourner à une intersection, comme celle-ci ne sera reconnue qu'après être passée devant, il faudra faire effectuer un demi-tour par le robot pour tourner dans cette intersection, ce qui n'est pas très naturel.

3.7 Conclusion

Nous avons présenté dans ce chapitre une adaptation des Processus Décisionnels de Markov à la robotique. Si celle-ci, au vu des approches existantes, n'est pas fondamentalement originale, elle a de notre point de vue le mérite de s'attacher à bien prendre en compte notre application de robotique mobile, ce qui n'est pas toujours le cas des approches similaires. La petite étude proposée des deux algorithmes principaux de résolution de MDP a permis de mettre en évidence leur différence de fonctionnement et a motivé l'étude de nouvelles méthodes permettant de converger plus rapidement vers la politique optimale.

En effet, parmi les travaux utilisant les Processus Décisionnels de Markov en robotique mobile, certains ne s'en servent que pour modéliser l'environnement et comme aide à la localisation, utilisant d'autres techniques de planification. La raison en est bien évidemment la complexité des algorithmes classiques de résolution de MDP. Mais vu l'intérêt des politiques générées par un bon paramétrage du modèle, il nous semble dommage de se passer de ces algorithmes. Plutôt que de tenter d'améliorer notre architecture d'exécution comme cela aurait pu être fait (apprentissage de la fonction d'observation), nous avons trouvé plus intéressant et plus novateur de nous intéresser aux techniques permettant d'obtenir plus rapidement des politiques sous-optimales mais néanmoins intéressantes. Les chapitres suivants sont consacrés à ces recherches.

Chapitre 4

Agrégation d'états

4.1 Introduction

Nous avons vu dans le chapitre précédent qu'une approximation de la politique optimale peut être obtenue par l'utilisation d'une politique *plus court chemin* classique, mais que l'exécution de celle-ci n'induira pas les mêmes comportements pour notre robot mobile. Nous avons donc cherché d'autres méthodes permettant d'obtenir rapidement une « bonne » politique. Parmi celles-ci, on distingue la décomposition d'environnement (dont nous parlerons au chapitre suivant), la construction de politiques partielles et l'agrégation d'actions ou d'états. Cette dernière technique constitue le sujet principal de ce chapitre. Les principes en sont simples : parmi les états servant à décrire l'environnement, certains ont des caractéristiques communes, qu'on peut exploiter pour réduire la taille du problème à résoudre. L'idée est de regrouper ces états *de base* au sein d'un état *abstrait*, chargé de représenter fidèlement les états de base qu'il contient. Au lieu de résoudre le Processus Décisionnel de Markov de base, on résout le MDP abstrait. L'intérêt est de réduire le plus possible l'espace d'états, afin d'accélérer la résolution du MDP. Idéalement, la solution du MDP abstrait pourra être utilisée directement sur le MDP de base : dans ce cas, l'action optimale définie pour un état abstrait sera exécutée en tous les états de base le constituant. Mais généralement la solution du MDP abstrait devra être transformée afin d'être applicable aux états de base.

La difficulté majeure consiste à détecter les caractéristiques importantes : qu'est ce qui fait qu'un groupe d'états peut être constitué, et surtout quelle influence cela aura-t-il sur la qualité de la politique obtenue ? Plusieurs approches ont été développées dans ce domaine. Après avoir présenté les plus significatives, nous introduisons la nôtre. Celle-ci utilise les caractéristiques de notre application et notamment le fait que les environnements considérés contiennent des couloirs, qu'il n'est pas forcément utile de représenter très fidèlement. En partant de cette hypothèse, nous présentons d'abord trois méthodes d'agrégation d'états, chacune essayant de pallier aux inconvénients de la précédente. Nous présentons les résultats obtenus aussi bien en terme de qualité des politiques générées qu'en terme de temps de calcul. Ces travaux nous ont permis de mettre en évidence un fait qui est peu utilisé habituellement dans les approches similaires : quelle que soit la qualité de la politique, son exécution ne peut se faire qu'à l'aide d'un module de navigation. Celui-ci

est utilisé afin d'éviter les obstacles inconnus lors de la phase de planification, mais aussi pour éviter les collisions dues à une mauvaise localisation du robot. Ceci nous a amené à développer une quatrième méthode d'agrégation, partant du principe que ce qui peut être géré par le module de navigation peut être négligé par la politique. Les politiques générées par cette méthode sont de mauvaise qualité, mais leur exécution est sous certaines conditions au moins aussi efficace que celle des politiques optimales.

4.2 Approches similaires

Avant de présenter spécifiquement les travaux utilisant des techniques d'agrégation, nous présentons rapidement les travaux cherchant à obtenir une politique partielle.

4.2.1 Résolution de MDP partiels

Ces travaux, qui ne relèvent pas de l'agrégation, sont néanmoins présentés ici car ils sont souvent comparés à ces derniers. L'idée est également de réduire l'espace de recherche de la solution, mais d'une façon différente. Au lieu de considérer tout l'espace mais de façon moins précise, on peut construire un MDP partiel ne contenant qu'un sous-ensemble des états du MDP de base. Pour résoudre un problème donné, on fait l'hypothèse que seuls un certain nombre d'états seront concernés, et les autres peuvent donc être négligés. Cette approche est développée dans [Dean *et al.*, 1993; Dean *et al.*, 1995], sur une application de navigation d'un robot mobile. L'état initial du robot et son but étant connus, un algorithme simple permet d'obtenir très rapidement un chemin reliant ces deux états. Cette politique est immédiatement envoyée au robot, qui commence à l'exécuter. Les états adjacents aux états du MDP partiel (contenant les états sur le chemin calculé) se voient affecter des actions-réflexes qui permettent de revenir sur le chemin si une mauvaise exécution d'action amène le robot à le quitter. Puis un second MDP partiel est construit, contenant les états du chemin auxquels s'ajoutent les états adjacents. On recalcule la politique optimale pour ce MDP, et on l'envoie au robot qui abandonne la politique précédente. Le processus continue ainsi jusqu'à ce que le robot ait rempli sa mission ou que le MDP partiel englobe tous les états du MDP de base. Cette approche permet une utilisation temps-réel des MDP, mais présente aussi des inconvénients. En effet, la rapidité d'exécution de la mission dépend fortement de l'algorithme chargé de trouver rapidement la première politique. Si celui-ci préconise un chemin très peu similaire à la politique stochastique optimale, le robot devra éventuellement faire demi-tour au cours de l'exécution de sa mission, ce qui peut faire perdre le bénéfice du gain de temps initial et augmente les risques de collision ou de désorientation du robot.

4.2.2 Méthodes d'agrégation

Travaux de *Boutilier* et *Dearden*

Une méthode générale d'agrégation d'états est donnée dans [Boutilier et Dearden, 1994; Dearden et Boutilier, 1997]. Tout d'abord, une représentation particulière des MDP

est donnée. On est ici plus proche du monde des blocs et de STRIPS que des représentations habituelles des MDP, dans lesquels l'espace d'états est constitué par les positions (géographiques) possibles dans l'environnement. Ici, les états sont des propositions logiques et les actions ne peuvent être exécutées que si certaines préconditions sont vérifiées. Ces actions ont bien entendu une influence sur le changement d'état, celui-ci étant probabiliste. Dans l'exemple donné, un robot doit aller chercher un café à un humain, tout en se protégeant de la pluie. On a ici une action *Move* qui lui permet de passer du laboratoire au café et inversement, mais le chemin entre les deux endroits n'est pas du tout planifié. Cette approche est donc assez éloignée de la nôtre, mais elle est tout de même très intéressante puisque les auteurs montrent comment le MDP de base est transformé en MDP abstrait, dans lequel seules les caractéristiques importantes sont conservées. Celles-ci sont déterminées automatiquement à l'aide de la fonction de gain. Les fonctions de gain et de transition sont ensuite modifiées pour prendre en compte le nouvel espace d'états et le MDP abstrait est résolu classiquement. La solution obtenue est directement applicable au modèle initial, et peut être utilisée pour obtenir la solution optimale.

Agrégation d'actions

D'autres travaux utilisent l'espace d'états sans le modifier mais essayent de planifier à plusieurs niveaux d'abstraction. C'est le cas notamment des travaux de *Tash* [Tash, 1994]. L'idée est qu'une action peut être prévue sans qu'on sache exactement comment elle sera exécutée, à condition qu'on connaisse ses effets probables. Au lieu de spécifier précisément pour chaque état une action optimale, on peut assigner des sous-buts à atteindre en fonction de la position courante. Une politique peut alors prendre la forme {*Aller de S en X; Aller de X au but*}. Au départ, on ne sait pas comment (par quel chemin) passer de S en X, mais on sait que cela réussira avec une probabilité p et qu'on atteindra Y au lieu de X avec une probabilité q. L'exécution de chaque action abstraite a également un coût, et la politique choisie devra faire un compromis entre le coût et la probabilité de succès de chaque action abstraite. L'intérêt est ici d'obtenir rapidement un plan, qui sera raffiné au fur et à mesure qu'on exécute chaque action abstraite.

Les travaux de *Haddawy* et *Doan* utilisent également l'agrégation d'actions [Haddawy *et al.*, 1995]. Dans cette approche, la représentation du monde est également plus proche du monde des blocs que de la représentation classique des MDP. Un état est ici une spécification de la situation du monde au temps t, les actions sont définies en fonction des préconditions qui doivent être remplies avant leur exécution, par les effets qu'elles ont sur le monde et par les probabilités correspondant aux différents effets. Une action abstraite peut être de deux types :

- Une concaténation de n actions de base : dans ce cas l'exécution d'une telle action consiste en l'exécution séquentielle des actions de base ;
- Un regroupement d'actions similaires. Dans ce cas, les préconditions et effets probables de l'action abstraite sont calculés en fonction de ceux des différentes actions de base correspondantes.

Le module de planification DRIPS *Decision-Theoretic Refinement Planning System* part d'un ensemble de plans au plus haut niveau d'abstraction. A chaque plan est assigné un

intervalle de valeur représentant son « utilité », et seuls les plans les plus utiles sont instanciés, ce qui permet de réduire l'espace de recherche.

4.2.3 Conclusion

Les travaux utilisant l'agrégation et l'abstraction le font soit au niveau des états, soit au niveau des actions. Ces deux approches sont tout à fait applicables à notre problème de planification en intérieur structuré. Une agrégation d'actions toute simple, consistant à l'exécution séquentielle de 10 actions `Avancer` pourrait permettre de trouver rapidement une bonne politique, surtout pour les environnements de grande taille dans lesquels un état peut être situé très loin du but, et est donc peu sensible à la récompense obtenue au but. Néanmoins, cela nécessiterait le recalcul des fonctions de transition, et surtout cela ne réduit pas l'espace d'états. Or la complexité de *Policy Iteration* est surtout fonction de cet espace, c'est donc surtout à ce niveau que l'agrégation nous semble prometteuse. De plus, l'application qui nous intéresse, comme nous le verrons au paragraphe suivant, se prête naturellement à l'agrégation d'états. C'est donc vers ce type d'agrégation que nous nous sommes tournés.

4.3 Notre approche

4.3.1 Généralités

Dans un environnement intérieur structuré comme le Loria, les missions d'un robot consistent généralement à emprunter divers couloirs, pour atteindre un endroit précis du laboratoire (bureau, intersection, porte, etc.) Imaginons notre robot placé dans l'environnement représenté par la figure 4.1. Pour atteindre son but (matérialisé par la croix), il devra suivre la politique optimale représentée figure 4.2.

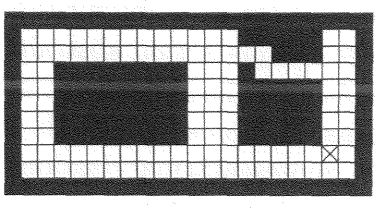


FIG. 4.1 – *Environnement simple non agrégé.*

La politique présentée ici affecte une action optimale à chaque état de l'environnement, classiquement représenté par 4 états pour chaque case de la grille. Mais ce plan peut aisément être représenté sous une autre forme. Le robot placé dans le coin en haut à gauche de l'environnement, la politique optimale pourrait être exprimée ainsi :

- Orientation vers l'est ;
- Traverser le couloir jusqu'à la prochaine intersection ;
- Pivoter vers la droite ;
- Traverser le couloir jusqu'au bout ;

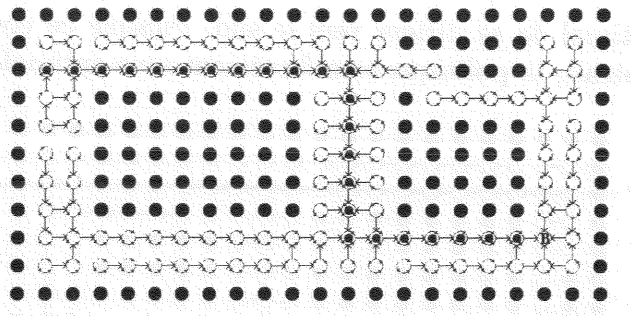


FIG. 4.2 – *Politique optimale de l'environnement de la figure 4.1.*

- Pivoter vers la gauche ;
- Traverser le couloir jusqu'au but (ou jusqu'à la prochaine intersection).

Cette reformulation de la politique optimale permet de distinguer deux types d'états. D'une part les états constituant les couloirs, et d'autre part les états situés aux intersections. Dans un couloir, la tâche à réaliser consiste à traverser celui-ci dans un sens ou un autre, jusqu'à atteindre une intersection, une porte ou le but de l'environnement. Au sein d'une intersection, le robot peut soit continuer son chemin, soit pivoter en vue d'emprunter un nouveau couloir. De ce fait, notre environnement de départ peut donner lieu à l'environnement de la figure 4.3, dans lequel les états constituant un couloir sont agrégés (formant ainsi 7 couloirs). La politique optimale peut alors être *approximativement* résumée par la figure 4.4.

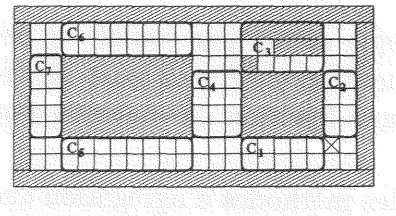


FIG. 4.3 – *Environnement après agrégation.*

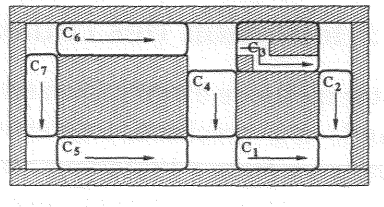


FIG. 4.4 – *Politique de l'environnement de la figure 4.3.*

Ces deux dernières figures appellent les remarques suivantes :

- La politique obtenue ne sera qu'une approximation de la politique optimale. L'affectation d'une seule action par couloir n'est pas toujours suffisante. Prenons par exemple le couloir vertical le plus à gauche, noté C_7. On peut voir sur la figure 4.3 que l'action optimale de certains états mène au nord, alors qu'en partant d'autres

états, il est optimal de se diriger vers le sud. Puisque ce couloir donne lieu à un unique état abstrait, la politique sera forcément sous-optimale.

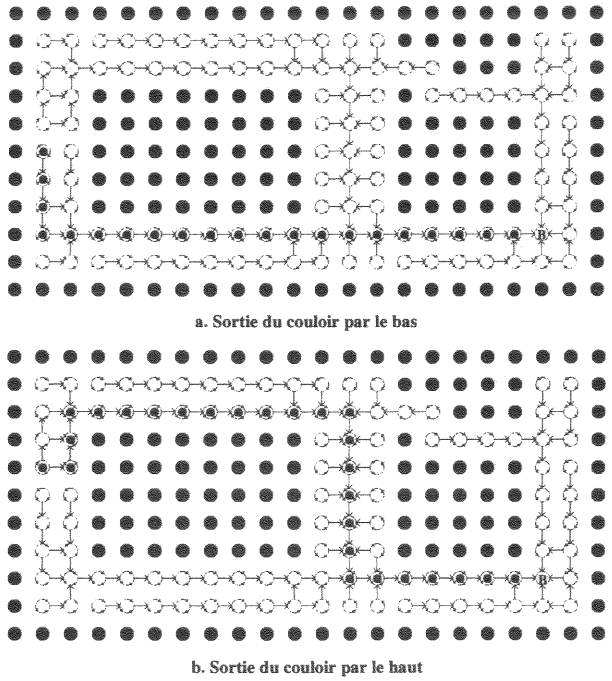


FIG. 4.5 – *1 couloir, 2 actions : traverser soit vers le haut, soit vers le bas.*

- L'affectation d'une unique action par couloir ne permet pas d'éviter les obstacles qui les obstruent éventuellement. Après avoir résolu le MDP abstrait, il sera donc nécessaire d'appliquer une fonction d'évitement d'obstacle rapide à chaque couloir qui en comporte.

Avant de voir plus en détail les méthodes d'agrégation que nous avons développées, nous explicitons dans le paragraphe suivant ce que nous appelons « couloir » et comment nous les différencions par rapport aux autres états de l'environnement.

4.3.2 Prétraitement : définition des couloirs

Nous venons de le dire, notre méthode d'agrégation est fondée sur les notions de couloir et d'intersection. Ceci demande des précisions : qu'entendons-nous par « couloir » et « intersection », et puisque seuls les états à l'intérieur des couloirs sont agrégés, comment distinguer les états d'un couloir des états constituant une intersection. Un couloir pourrait être défini comme une portion de l'environnement plus ou moins rectiligne et cernée par des obstacles. En fait, cette notion est assez difficile à définir et il est encore plus difficile de décider par un processus automatique si tel état fait partie d'un couloir ou au contraire d'une intersection. Des travaux Ce prétraitement nécessaire est donc réalisé manuellement sous la forme d'un fichier décrivant l'environnement, comme on peut le voir figure 4.6. Chaque position de l'environnement est étiquetée :

- '1' : la position est (partiellement ou non) occupée par un obstacle ;

```
MAP
11111111111111111111
1..hhhhhhhh...11111..1
1..hhhhhhhh...hh111..1
1vv11111111...1hhhh..1
1vv11111111vvv11111vv1
1vv11111111vvv11111vv1
1vv11111111vvv11111vv1
1vv11111111vvv11111vv1
1..hhhhhhhh...hhhhh..1
1..hhhhhhhh...hhhhh..1
11111111111111111111
ENDMAP
```

FIG. 4.6 – *Fichier de description de l'environnement de la figure 4.1.*

- 'h' : la position fait partie d'un couloir horizontal ;
- 'v' : la position fait partie d'un couloir vertical ;
- '.' : la position est libre, elle fait partie d'une intersection ou du but.

Il est à noter que si le but à atteindre est placé au milieu d'un couloir, celui-ci est coupé en deux et séparé par une intersection qui contient le but à atteindre.

4.3.3 En résumé

Notre technique d'agrégation se résume en quatre points :

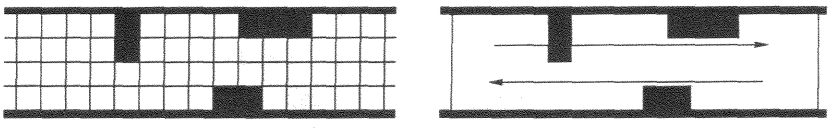
1. Transformation du MDP de base en MDP abstrait, en agrégeant les états constituant les couloirs ;
2. Résolution du MDP abstrait, constitué d'états abstraits (les couloirs) et d'états de base (les intersections) ;
3. Transfert de la politique trouvée pour le MDP abstrait au MDP de base ;
4. Retraitement des états situés à l'intérieur des couloirs pour éviter les obstacles éventuels.

Nous venons de présenter les bases de notre technique d'agrégation. Dans les paragraphes suivant, nous présentons trois méthodes d'agrégation dont le point commun est l'utilisation des couloirs pour la création d'états abstraits, mais qui différent au niveau de la représentation de ces couloirs.

4.4 Première méthode d'agrégation

Dans ce paragraphe, nous présentons la première méthode d'agrégation que nous avons développée. Nous partons ici du principe que quand le robot se trouve dans un couloir, peu importe la façon qu'il a de le traverser, ce qui compte c'est la direction qu'il prend : soit elle le rapproche du but, soit elle l'éloigne. Dans un couloir horizontal, il s'agit soit de

se diriger vers l'est, soit de se diriger vers l'ouest. En revanche, dans un couloir vertical, seules les directions nord et sud sont intéressantes. Chaque couloir va donc donner lieu à deux états abstraits, dans le sens du couloir, ce qui est représenté par la figure 4.7. Un couloir horizontal est représenté à l'aide de deux états, l'un orienté vers l'est, l'autre orienté vers l'ouest ; un couloir vertical est représenté par un état orienté au nord, et un état orienté au sud.



a. Sans agrégation : 212 états. **b. Après agrégation : 2 états.**

FIG. 4.7 – *Première méthode d'agrégation.*

Les couloirs étant ainsi agrégés, les actions du MDP de base n'ont plus aucun sens dans notre MDP abstrait. Puisqu'il s'agit de traverser les couloirs dans un sens ou dans un autre (sans se préoccuper de la façon dont le couloir est traversé), nous introduisons deux nouvelles actions :

- `Traverser_Couloir` : cette action permet de passer de l'intérieur du couloir aux états adjacents, situés à proximité ;
- `Demi_Tour` : ceci permet de changer de direction : si le robot est orienté dans la mauvaise direction, cette action lui permettra de pivoter pour traverser le couloir dans l'autre sens.

Pour mettre en œuvre ces deux nouvelles actions, il sera bien entendu nécessaire de connaître la fonction de transition correspondante. Mais avant de voir comment nous allons générer cette fonction, nous nous intéressons à un autre problème (d'ailleurs lié à celui-ci), qui est celui consistant à conserver les caractéristiques importantes des couloirs lors du passage au MDP abstrait.

4.4.1 Caractéristiques importantes des couloirs

Puisque chaque couloir donne uniquement lieu à deux états, l'espace d'états est considérablement réduit, ce qui permet d'envisager des temps de calcul très rapides. Mais ceci n'est bien entendu intéressant que si les politiques obtenues ont un sens, c'est-à-dire que si elles sont « proches » des politiques optimales obtenues sur les environnements non agrégés. La transformation de 212 états de base en 2 états abstraits (comme sur notre exemple de la figure 4.7) n'a de sens que si les caractéristiques des couloirs sont conservées lors du passage des états de base aux états abstraits. Dans ce paragraphe, nous montrons l'importance des obstacles et de la longueur des couloirs.

Importance de la longueur des couloirs

Lorsqu'il est possible d'atteindre un but en passant par plusieurs chemins différents, il semble évident que la longueur des couloirs à emprunter va influer sur le choix d'un chemin par rapport à un autre. Prenons par exemple l'environnement de la figure 4.8.

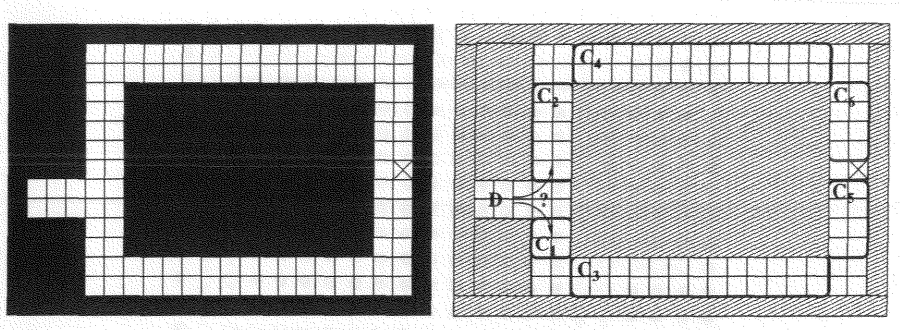


FIG. 4.8 – *Influence de la longueur des couloirs.*

Partant du point marqué 'D' sur la figure, le but (matérialisé par une croix) peut être atteint soit en passant par le couloir du haut (C_4), soit via celui du bas (C_3), qui ont la même longueur. Mais pour atteindre ces couloirs, il faut d'abord emprunter soit le couloir C_1, soit C_2, qui eux, n'ont pas la même longueur. Comme on peut le voir sur la politique optimale représentée par la figure 4.9, il est plus intéressant d'emprunter C_1, car ce couloir est plus court que C_2. Ainsi, si nous voulons que les politiques calculées sur les MDP abstraits soient proches de l'optimal, il est indispensable que les états abstraits représentant les couloirs prennent en compte la longueur du couloir.

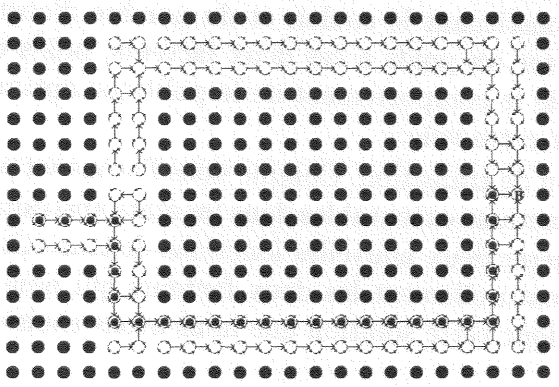


FIG. 4.9 – *Politique optimale pour l'environnement de la figure 4.8.*

Présence d'obstacles

L'autre caractéristique à prendre en compte est bien entendu la présence éventuelle d'obstacles. Dans le même environnement que celui de la figure 4.8, nous avons ajouté des obstacles dans le couloir horizontal du bas (C_3). On peut voir sur la figure 4.10 que dans ce cas, la politique optimale préconise d'emprunter C_2, qui permet de passer par le couloir du haut, qui n'est pas obstrué. Ce chemin, bien que plus long en terme de distance, est meilleur au sens de la politique optimale. Il est donc primordial que la présence ou non d'obstacles (et leur nombre) au sein d'un couloir soit représentée par l'état abstrait.

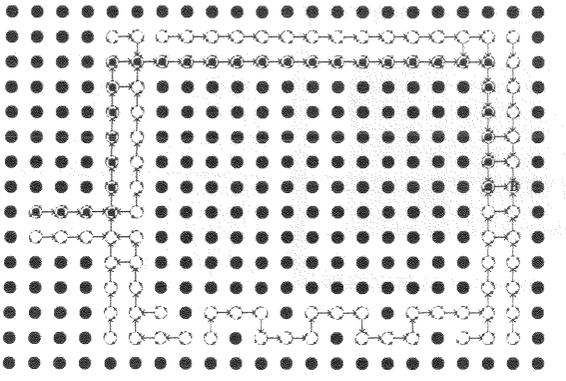


FIG. 4.10 – *Politique optimale pour la figure 4.8 dans laquelle on a ajouté des obstacles.*

4.4.2 Prise en compte de ces caractéristiques

Nous venons de le voir, la manière dont nous allons représenter les états abstraits faisant office de couloirs doit prendre en compte leur longueur et les obstacles qu'ils contiennent. Pour cela, nous avons envisagé deux possibilités :

- Utiliser la fonction de gain, qui affecte une récompense au fait de se trouver en un état. Plus un couloir est long et obstrué, moins il est intéressant de l'emprunter. La fonction de gain pourrait donc être générée automatiquement en fonction du nombre d'états constituant le couloir et du nombre d'obstacles. Le problème à résoudre est d'affecter un poids à ces deux facteurs, poids qui représenterait le plus fidèlement possible le comportement de le politique optimale. Cette difficulté nous a fait abandonner cette voie.
- Utiliser la fonction de transition. Comme nous l'avons vu précédemment, l'introduction de nouvelles actions exécutables dans les états abstraits représentant les couloirs nous oblige à générer deux nouvelles fonctions de transition. Il se trouve que celles-ci vont intégrer naturellement les deux facteurs qui nous intéressent : longueur et obstacles. C'est donc cette solution que nous avons choisie.

4.4.3 Génération des fonctions de transition

Nous l'avons vu précédemment : placé au milieu d'un couloir, le robot a pour mission de le traverser pour en sortir. Pour ce faire, nous avons introduit deux nouvelles actions, uniquement exécutables (et à l'exclusion des autres) dans les états abstraits. Depuis un état de base compris dans un couloir, il est possible d'atteindre soit une intersection ou un autre couloir (si le robot est situé à proximité de l'une des extrémités du couloir), soit un autre état de ce couloir ou un obstacle (mur délimitant le couloir ou obstacle situé à l'intérieur du couloir). Partant du principe qu'à l'intérieur d'un couloir, tous les états de base sont équiprobables, la fonction de transition pour l'action consistant à traverser le couloir c est définie ainsi :

- Vers un état de base s' (état situé à une intersection) :

$$T_c(c,\texttt{Traverser\_couloir},s') = \frac{\sum_{s\in c} T(s,\texttt{Avancer},s')}{|c|}$$

- Probabilité de collision :

$$T_c(c,\texttt{Traverser\_couloir},s_o) = \frac{\sum_{s\in c}\sum_{s'\in \texttt{suiv\_obs}(s)} T(s,\texttt{Avancer},s')}{|c|}$$

- Vers un autre couloir c' :

$$T_c(c,\texttt{Traverser\_couloir},c') = \frac{\sum_{s\in c}\sum_{s'\in c' \cap \texttt{suiv\_libre}(s)} T(s,\texttt{Avancer},s')}{|c|}$$

- La probabilité d'auto-transition est obtenue en posant $c' = c$ dans la formule précédente.

L'ensemble $\texttt{suiv\_libre}(s)$ est constitué par les états non obstrués s' qu'il est possible d'atteindre en exécutant une action `Avancer` depuis l'état s ; l'ensemble $\texttt{suiv\_obs}(s)$ contient quant à lui les états obstrués par un obstacle pouvant être atteints depuis s en exécutant une action `Avancer`.

Il est à noter que cette fonction de transition permet de représenter efficacement les caractéristiques du couloir. Plus le couloir est long, plus la probabilité d'auto-transition est forte, et plus la probabilité d'en sortir est faible. Ainsi, un couloir long aura une valeur faible et sera donc moins emprunté qu'un couloir plus court. De façon similaire, un couloir comportant beaucoup d'obstacles mènera avec une forte probabilité à une collision. Ceci rendra également la valeur de l'état abstrait faible, et le couloir sera moins intéressant à emprunter. Ainsi, les deux caractéristiques importantes sont représentées par la fonction de transition, ce qui nous permet de laisser inchangée la fonction de gain.

La probabilité d'atteindre un couloir est définie simplement comme la somme des probabilités d'atteindre un des états de base constituant le couloir. Les actions de pivotement n'opérant aucune translation, il n'est possible d'atteindre un couloir qu'en exécutant une action `Avancer` :

$$T(s,\texttt{Avancer},c) = \sum_{s'\in c \cap \texttt{suiv\_libre(s)}} T(s,\texttt{Avancer},s')$$

Les transitions pour l'action `Demi_tour` sont calculées en appliquant deux actions successives de pivotement : un couloir a alors pour successeur l'état représentant ce même couloir mais dans le sens inverse (probabilité 0,86) et une auto-transition de probabilité 0,14.

4.4.4 Transformation de la politique obtenue

Une fois la fonction de transition recalculée pour les états abstraits, il ne reste plus qu'à résoudre classiquement le MDP. Après quelques itérations de *Policy Iteration*, nous obtenons une politique qu'il faut adapter au MDP de base. Pour les états hors-couloir (qui n'ont pas donné lieu à un état agrégé), le transfert est immédiat : l'action optimale dans le MDP de base est la même que dans le MDP abstrait. En revanche, les états de base d'un couloir doivent se voir affecter une action optimale. Grossièrement, tout état de base d'un état abstrait dont l'action optimale est `Traverser_Couloir` se voit affecter une action

`Avancer`, alors que l'action `Demi_tour` est transformée en deux actions `Pivoter_gauche` successives. Mais ce n'est pas toujours vrai :

- Un état abstrait regroupe plusieurs orientations possibles du robot au sein du couloir. De ce fait, l'action `Avancer` est affectée aux états « dans le sens » du couloir, et des actions de pivotement sont affectées aux états orientés orthogonalement.
- Certains couloirs contenant des obstacles, l'action optimale peut éventuellement amener à des collisions dans certains états de base. Une petite procédure de contournement d'obstacles doit donc être appliquée lors du transfert d'actions du MDP abstrait au MDP de base.

Nous ne donnerons pas ici l'algorithme que nous utilisons pour faire ce transfert d'actions, puisque les nombreux cas de collision à traiter le rendent très peu clair et peu propre. Qu'il suffise au lecteur de savoir que celui-ci permet d'éviter les collisions simples, mais que toute configuration en cul-de-sac lui est fatale. Néanmoins, sur les environnements de test, cet algorithme s'est avéré suffisant.

4.4.5 Intérêt de cette méthode

Les résultats obtenus avec cette première méthode sont présentés au paragraphe 4.7. Dans cette section, nous nous contentons de montrer les avantages et inconvénients de cette méthode sur les petits exemples précédents. Tout d'abord, les figures 4.11 et 4.12 montrent bien que les caractéristiques des couloirs sont efficacement prises en compte par notre fonction de transition. Sur ces figures, la politique optimale calculée sur l'environnement de base est située à gauche, alors que celle calculée sur l'environnement agrégé est à droite. Sur les politiques des environnements agrégés, les actions `Transformer_Couloir` et `Demi_Tour` affectées aux états abstraits ont été transformées comme dit précédemment.

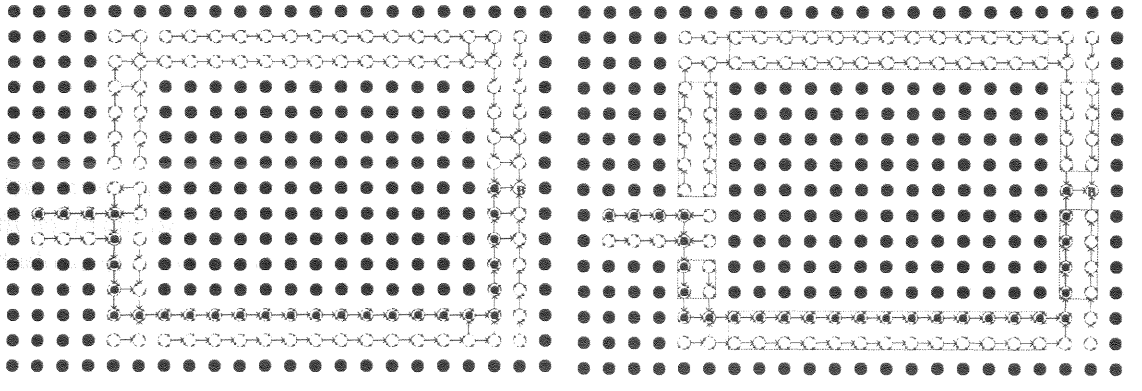


FIG. 4.11 – *Politique optimale (1) environnement de base (2) environnement agrégé.*

Sur les deux figures, le chemin choisi pour atteindre le but est identique selon la politique choisie. Dans le premier cas (sans obstacle), la politique agrégée représente 99,48% de la politique optimale ; ce chiffre est de 99,21% dans le second (avec obstacles). On note évidemment quelques différences, mais rien de très surprenant. A ce stade, sur ces petits environnements, notre méthode d'agrégation semble très satisfaisante. Mais le résultat obtenu sur l'environnement de la figure 4.1 (page 68) est moins encourageant, comme on

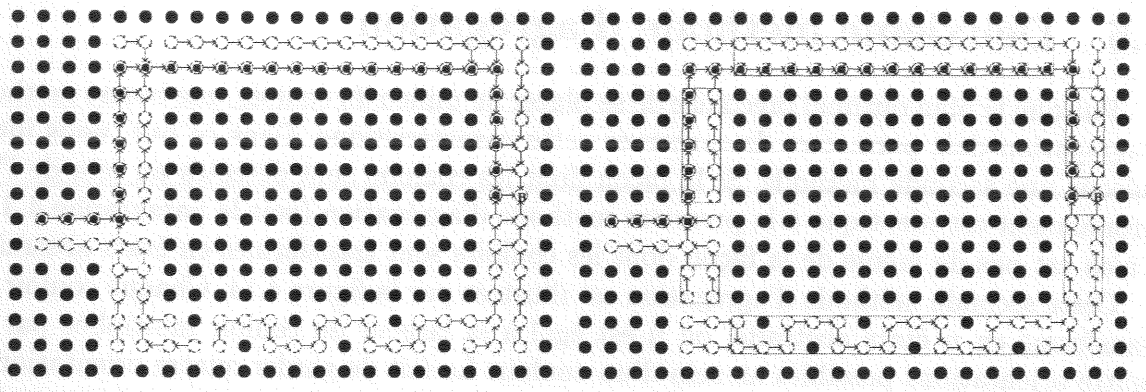


FIG. 4.12 – *Politique optimale (1) environnement de base (2) environnement agrégé.*

peut le voir sur la figure 4.13. Le premier problème, qui concerne le couloir vertical de gauche, n'est pas étonnant puisqu'il est la conséquence directe de notre façon d'agréger les états. En effet, dans ce couloir la politique optimale mène soit vers le nord, soit vers le sud, selon l'état initial dans lequel se trouve le robot. Ce comportement n'est pas retrouvé dans la politique de l'environnement agrégé, puisque une seule action optimale est affectée à chaque couloir. Ce problème est donc inévitable, à moins de découper plus finement les couloirs. Le second problème concerne le couloir vertical central. On s'aperçoit que la politique optimale mène le robot au centre du couloir, afin d'éviter de longer les murs, ce qui peut provoquer des collisions. Notre méthode d'agrégation ne prend absolument pas en compte cette caractéristique, ce qui entraîne une grande différence entre les deux politiques. Ici, la politique de l'environnement agrégé ne représente que 94,8% de la politique optimale, ce qui est un résultat faible pour un petit environnement comme celui-ci.

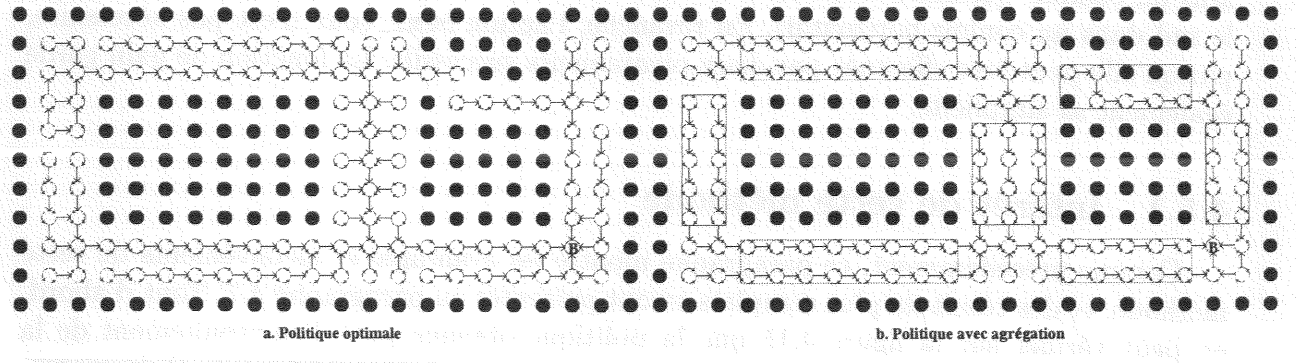


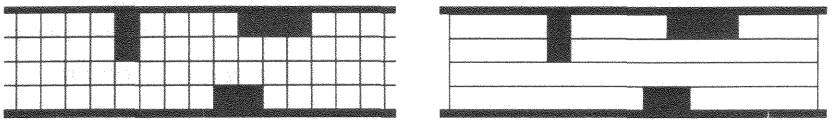
FIG. 4.13 – *Problème : pas de prise en compte de la largeur.*

A la lumière de ces petits exemples, il apparaît que notre méthode est bien adaptée aux environnements dont les couloirs sont étroits (largeur inférieure ou égale à deux états de base), mais qu'elle ne permet pas de gérer efficacement les environnements aux couloirs plus larges (largeur supérieure à deux états de base). C'est pour régler ce problème que nous avons développé la seconde méthode d'agrégation, exposée dans le paragraphe suivant.

4.5 Deuxième méthode d'agrégation

Puisque le plan optimal préconise de se placer au milieu du couloir, une bonne méthode d'agrégation doit permettre de conserver cette caractéristique. Dans ce but, nous introduisons une seconde méthode. Si la façon dont les couloirs sont détectés au sein de l'environnement est identique à la première méthode d'agrégation (voir le paragraphe 4.3.2), il n'en est pas de même de la façon dont les couloirs sont transformés en états abstraits.

Cette seconde méthode d'agrégation est illustrée par la figure 4.14. Chaque couloir est transformé en n états abstraits, où n est égal à la largeur du couloir multipliée par les quatre orientations possibles au sein d'un couloir. Le fait de garder quatre orientations a l'inconvénient de limiter la réduction de l'espace d'états, mais a l'avantage de permettre les comportements de navigation au milieu des couloirs, ce qui améliore la qualité des politiques obtenues. Dans notre exemple, on passe donc de 212 à 16 états (quatre couloirs dans lesquels quatre orientations sont possibles).



a. Sans agrégation : 212 états. b. Après agrégation : 16 états.

FIG. 4.14 – *Seconde méthode d'agrégation.*

Les caractéristiques importantes des couloirs mises en évidence précédemment, à savoir leur longueur et les obstacles qu'ils peuvent éventuellement contenir, sont prises en compte de la même façon que précédemment, grâce à la fonction de transition. L'action `Traverser_couloir` a été conservée, mais elle n'est possible que dans les états "dans le sens du couloir" (orientés vers l'est ou l'ouest dans un couloir horizontal ; vers le nord ou le sud dans un couloir horizontal). En revanche, l'action `Demi_tour` n'est pas nécessaire, puisqu'elle peut être effectuée par deux pivotements successifs. La fonction de gain garde également sa simplicité.

4.5.1 Intérêt de cette méthode

Comme précédemment, ce paragraphe montre les avantages et inconvénients de cette méthode, et des résultats plus complets seront présentés au paragraphe 4.7. Tout d'abord, on peut vérifier sur la figure 4.15 que la politique obtenue pour l'environnement de la figure 4.1 corrige les erreurs de la méthode précédente dans le couloir vertical du centre. Dans ce couloir, la politique optimale de l'environnement agrégé est identique à celle générée pour l'environnement initial. Cette politique représente 99,24% de la politique optimale, ce qui est un bien meilleur résultat qu'avec la méthode précédente.

Mais cette méthode a également de petits inconvénients. On peut le voir sur la figure 4.15, mais c'est encore plus flagrant sur les figures 4.16 et 4.17. Sur ces environnements, les chemins préconisés sont identiques selon les politiques, mais dans les deux cas la valeur de la politique de l'environnement agrégé n'est que de 96,03% (alors qu'elle était de plus de 99% avec la première méthode). Cette baisse de qualité est due à un effet

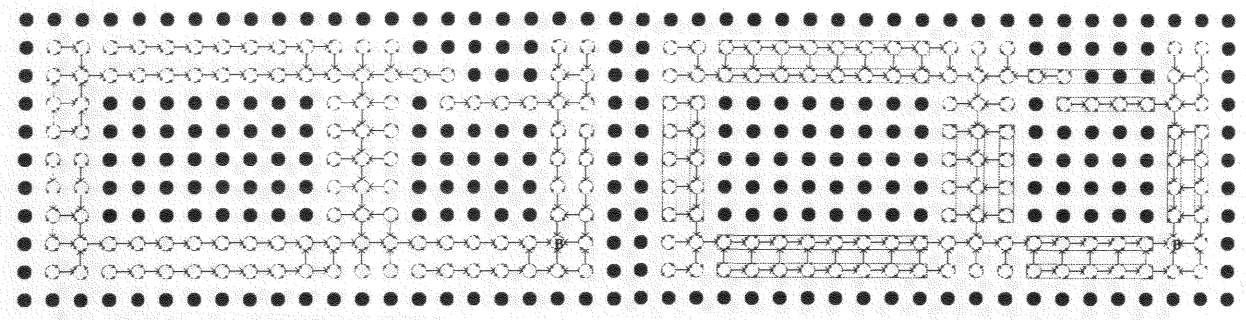


FIG. 4.15 – *Problème de largeur de couloir corrigé.*

indésirable de cette méthode. Sur la politique optimale de la figure 4.16, l'action optimale de la majorité des états du couloir horizontal du bas mène vers la droite. Seule la dernière position du couloir (indiquée par une flèche) mène à la fois vers le haut (si le robot est orienté au nord) et vers la droite (si le robot est orienté vers l'est). De ce fait, dans l'environnement agrégé la portion « haute » du couloir a une valeur plus intéressante que la portion « basse », ce qui explique pourquoi l'action optimale de l'état abstrait représentant la portion « basse » du couloir mène vers la portion « haute ». Cette même erreur est répétée dans les six couloirs, ce qui explique la baisse de qualité des politiques obtenues. On pourrait d'ailleurs s'étonner que cette erreur, également présente sur la figure 4.15, n'entraîne pas de baisse de qualité (au contraire, dans ce cas la qualité est meilleure). C'est tout simplement dû au fait que la correction d'erreur dans le couloir vertical central compense largement l'introduction des erreurs dans les cinq couloirs plus étroits.

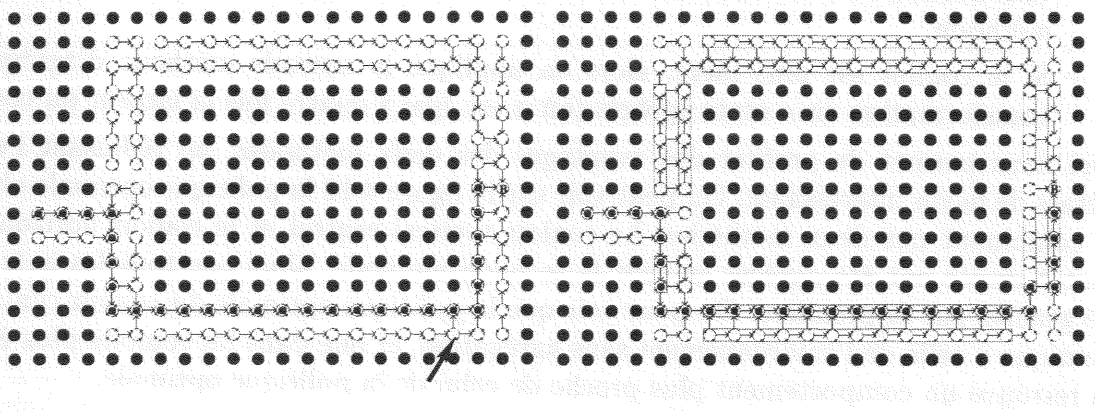


FIG. 4.16 – *Politique optimale (1) environnement de base (2) environnement agrégé.*

4.6 Intégration des deux méthodes

Nous venons de présenter deux méthodes d'agrégation, chacune avec ses avantages et ses inconvénients. En résumé, la première a l'avantage de réduire considérablement l'espace d'états, mais elle génère des politiques de piètre qualité lorsque les couloirs sont larges. La seconde réduit l'espace d'états de façon moindre, mais permet d'approcher de façon plus satisfaisante les politiques optimales en présence de larges couloirs, tout en introduisant

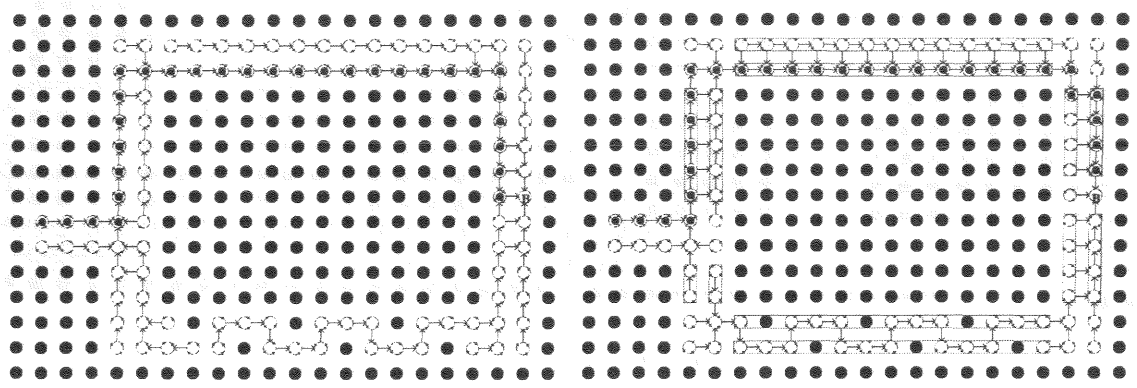


FIG. 4.17 – *Politique optimale (1) environnement de base (2) environnement agrégé.*

des petites erreurs dans les couloirs plus étroits. Les avantages de la première méthode correspondant aux inconvénients de la seconde, il devrait être intéressant de combiner les deux.

Comme précédemment, seuls les états situés dans les couloirs sont agrégés. Le prétraitement permettant de différencier les couloirs des autres états reste identique (voir le paragraphe 4.3.2). En revanche, la transformation des couloirs en états abstraits est modifiée, en utilisant un critère très simple :

- Si la largeur du couloir est inférieure ou égale à deux états de base, celui-ci est transformé en deux états abstraits auxquels il est possible d'appliquer soit une action `Traverser_couloir`, soit une action `Demi_Tour`, comme préconisé par la première méthode.
- Si le couloir est plus large, alors on applique la seconde méthode d'agrégation : transformation en n états agrégés (où n est égal à quatre fois la largeur du couloir).

Ceci nous permet de profiter des avantages des deux méthodes, en réduisant l'espace d'états de façon intéressante. On peut voir sur la figure 4.18 la politique optimale obtenue en combinant nos deux méthodes. Le couloir vertical central et le couloir horizontal en haut à droite ont été agrégés suivant la seconde méthode, alors que tous les autres ont été générés par la première. On peut voir que la politique optimale du couloir central est identique à celle de l'environnement sans agrégation, et que dans les couloirs plus étroits, on retrouve un comportement plus proche de celui de la politique optimale.

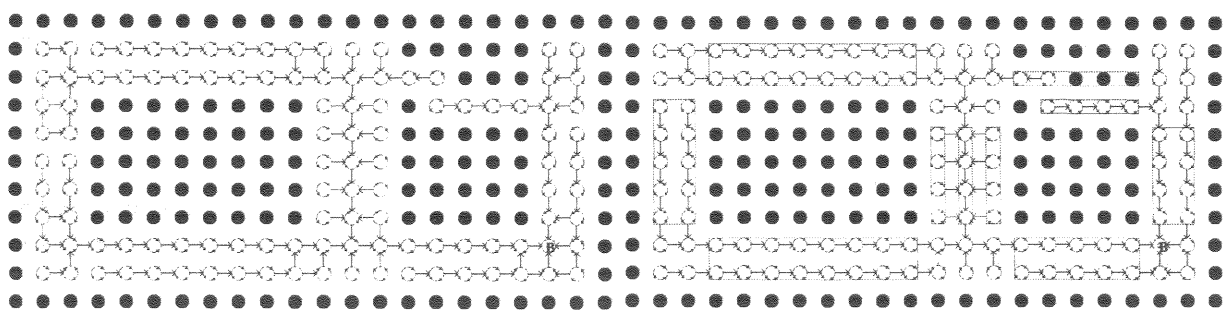


FIG. 4.18 – *Politique obtenue en combinant les deux méthodes.*

Pour résumer, la table 4.1 présente les résultats obtenus sur les petits exemples qui nous ont servi à illustrer nos méthodes. Sans entrer dans les détails, plusieurs remarques peuvent être faites :

- Tout d'abord, l'agrégation d'états permet de réduire assez sensiblement l'espace d'états, même sur des petits environnements comme ceux que nous avons présentés.
- Le nombre d'erreurs, qui correspond au nombre d'états pour lesquels l'action optimale calculée sur l'environnement agrégé est différente de l'action optimale de l'environnement de base, est assez grand pour la deuxième méthode, même quand la qualité de la politique est très bonne. Ce nombre d'erreurs est réduit lorsqu'on passe à la combinaison des deux méthodes, tout en augmentant très légèrement la qualité de la politique trouvée.

Mais ces environnements sont un peu trop simples pour nous permettre de tirer des conclusions quant aux mérites respectifs de nos méthodes. Le paragraphe suivant présente des résultats obtenus sur des environnements plus intéressants.

Figure 4.1	Optimal	Méthode 1	Méthode 2	Combinaison des 2
Nombre d'états	440	142	188	158
Valeur de la politique	-28589,79	-29384,03	-28707,20	-28702,97
% par rapport à l'optimal	100	94,85	99,24	99,27
Nombre d'erreurs	0	64	53	35
Figure 4.8	Optimal	Méthode 1	Méthode 2	Combinaison des 2
Nombre d'états	436	120	156	idem
Valeur de la politique	-33014,92	-33070,23	-33434,94	première
% par rapport à l'optimal	100	99,48	96,03	méthode
Nombre d'erreurs	0	27	67	
Figure 4.8 + obstacles	Optimal	Méthode 1	Méthode 2	Combinaison des 2
Nombre d'états	412	120	156	idem
Valeur de la politique	-31879,37	-31953,35	-32248,94	première
% par rapport à l'optimal	100	99,21	96,03	méthode
Nombre d'erreurs	0	41	72	

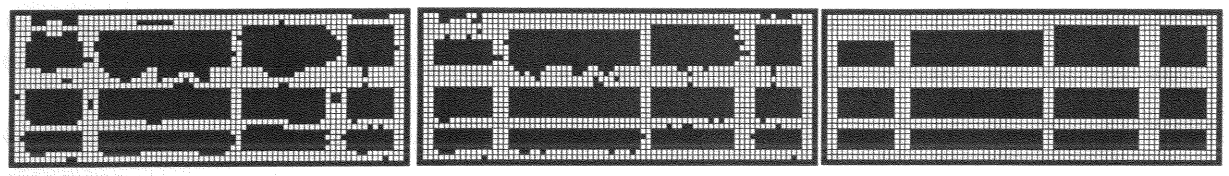
TAB. 4.1 – *Résultats des trois méthodes d'agrégation*

4.7 Résultats

Ce paragraphe présente les résultats obtenus sur nos environnements de test, figure 4.19 et 4.20. Nous examinons tout d'abord la qualité des politiques obtenues, puis nous présentons les temps de calcul avant de montrer la robustesse de nos méthodes en fonction de la présence ou non d'obstacles dans les environnements.

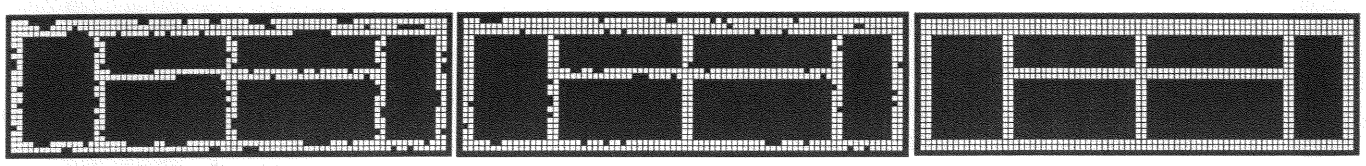
4.7.1 Qualité des solutions

Si les figures 4.19 et 4.20 ne présentent que trois instances de chaque environnement, ici nous en avons créé dix, afin d'avoir une idée plus précise de l'incidence des obstacles



a. 191 obstacles (3360 états) b. 81 obstacles (3800 états) c. Aucun obstacle (4124 états)

FIG. 4.19 – *Premier environnement de test.*



a. 130 obstacles (2192 états) b. 58 obstacles (2480 états) c. Aucun obstacle (2712 états)

FIG. 4.20 – *Deuxième environnement de test.*

sur notre approche : la première contient tous les obstacles, qui sont progressivement supprimés jusqu'à obtenir la dernière instance, qui en est totalement dénuée. Dans le premier environnement, le but a été successivement placé en 20 positions différentes ; dans le second, il a été déplacé 13 fois. Dans un premier temps, nous présentons les résultats globaux, c'est-à-dire sur 200 expériences dans le premier cas, 130 dans le second. Les résultats sont évalués en fonction des deux critères précédemment introduits : la qualité par rapport à la solution optimale et le pourcentage d'états pour lesquels l'action optimale de la solution agrégée est égale à l'action optimale de la solution de base (voir le paragraphe 3.4.1, page 40). Pour ces deux chiffres, on donne à chaque fois la valeur moyenne obtenue sur les 200 (ou 130) expériences, le meilleur résultat et le moins bon. De plus, afin d'avoir une idée plus précise de la qualité des solutions trouvées, nous les comparons aux solutions données par un algorithme *plus court chemin* (présenté au paragraphe 3.4.3, page 48).

Les tables 4.2 et 4.3 présentent les résultats obtenus pour les deux environnements précités. Trois tendances peuvent être dégagées de ces deux tables :

1. La première méthode est très peu satisfaisante. En effet, les résultats obtenus sont très proches voire inférieurs aux résultats obtenus par l'algorithme *plus court chemin*. La seule satisfaction que nous pouvons tirer de cette méthode vient du fait qu'elle ne descend jamais au-dessous de 39%, alors que l'algorithme *plus court chemin* présente un plus mauvais résultat de moins de 27%.
2. La seconde méthode s'est avérée être bien adaptée aux environnements de test. Dans les deux cas, le résultat moyen est bien supérieur au résultat de la première méthode (89,94% par rapport à 68,44% pour le premier environnement, et 94,87% contre 82,76% pour le second). Les plus mauvais résultats obtenus ont une valeur supérieure à 74% de l'optimal pour le premier environnement, et de presque 78% pour le second, ce qui est également satisfaisant. De plus, cette méthode permet dans certains cas d'atteindre jusqu'à 98% de l'optimal. Les résultats globaux sont jugés satisfaisants, tout comme le pourcentage d'actions correctes, qui est bien au-dessus de celui donné par l'algorithme *plus court chemin*.
3. La combinaison des deux méthodes permet d'améliorer légèrement les résultats. En

fait, ce sont surtout les états situés loin du but qui bénéficient de cette méthode, puisque si la qualité varie peu, en revanche le pourcentage d'actions correctes augmente plus sensiblement.

	Méthode 1	Méthode 2	Combinaison des 2	Solution *plus court chemin*
Qualité moyenne	68,44	89,94	90,21	65,05
Qualité minimum	39,94	74,95	74,83	26,78
Qualité maximum	90,02	97,90	98,06	88,87
% Action moyen	69,55	80,68	82,86	66,06
% Action minimum	62,46	75,75	77,90	53,73
% Action maximum	75,74	88,14	90,93	73,84

TAB. 4.2 – *Résultats sur l'environnement de la Figure 4.19*

	Méthode 1	Méthode 2	Combinaison des 2	Solution *plus court chemin*
Qualité moyenne	82,76	94,87	95,30	83,27
Qualité minimum	42,79	77,94	78,14	34,00
Qualité maximum	97,44	98,08	99,50	97,60
% Action moyen	73,45	82,47	83,46	74,88
% Action minimum	62,32	77,62	77,83	54,02
% Action maximum	80,29	88,24	93,14	85,54

TAB. 4.3 – *Résultats sur l'environnement de la Figure 4.20*

4.7.2 Traitement des bureaux, pièces, halls, etc.

Notre technique d'agrégation étant fondée sur la transformation de couloirs en états abstraits, il nous a semblé intéressant de la tester sur des environnements comportant également des pièces et bureaux. *A priori*, deux possibilités permettent d'utiliser nos algorithmes sur de tels environnements :

- On peut uniquement agréger les états situés réellement dans un couloir, et traiter les états composant les bureaux comme les états constituant les intersections : ils ne donnent lieu à aucune agrégation. Dans ce cas, les résultats devraient être similaires à ceux précédemment présentés, mais la réduction du temps de calcul sera bien moindre, puisque l'espace d'états sera réduit de façon moins conséquente.
- On peut considérer qu'un bureau n'est qu'un couloir un peu particulier. Après tout, il s'agira principalement d'en sortir ou d'y entrer. On peut alors agréger les états constituant un bureau de la même façon que s'il s'agissait d'un couloir, ce qui permettrait de réduire de façon importante l'espace d'états.

La seconde alternative nous ayant semblé plus intéressante, nous l'avons testée sur le petit environnement représenté par la figure 4.21. Les résultats obtenus sont moins intéressants que les précédents. En effet, les résultats de la deuxième et troisième méthode sont supérieurs à ceux obtenus par l'algorithme *plus court chemin*, mais selon la position du but, on peut obtenir de très mauvais résultats, puisque le plus mauvais score est inférieur à 50% de l'optimal. Ceci est dû au fait que seul un petit nombre d'états permet de sortir d'une pièce (états contigus à la porte) et de ce fait la probabilité d'auto-transition est beaucoup trop forte par rapport à la probabilité de sortir.

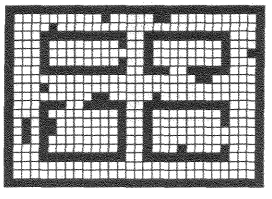


FIG. 4.21 – *Troisième environnement (1640 états).*

	Méthode 1	Méthode 2	Combinaison des 2	Solution *plus court chemin*
Qualité moyenne	74,25	87,32	87,32	75,83
Qualité minimum	40,21	47,44	47,43	51,21
Qualité maximum	87,25	97,87	98,03	86,89
% Action moyen	71,96	85,02	86,44	73,84
% Action minimum	66,20	82,52	82,64	67,30
% Action maximum	76,89	87,47	89,98	77,44

TAB. 4.4 – *Résultats pour l'environnement Figure 4.21*

4.7.3 Temps de calcul

Nous venons de voir que la qualité des politiques calculées en utilisant notre approche est globalement satisfaisante. Reste à voir le gain de temps que ces méthodes peuvent apporter. Leur complexité est assez difficile à exprimer : la résolution du MDP nécessite les mêmes opérations, mais sur un espace d'états et d'actions réduit. Le temps passé lors de la phase d'agrégation (construction de la nouvelle fonction de transition) dépend du nombre de couloirs traités et du nombre d'états dans ces couloirs, qui est variable. Les tables 4.5, 4.6 et 4.7 contiennent les temps de calcul moyens obtenus sur nos 3 environnements (356 problèmes au total). Ces valeurs sont des temps CPU exprimés en secondes, et ont été obtenus sur un quadri-processeur Sun Ultra Sparc. Le calcul de la solution optimale a été fait en initialisant *Policy Iteration* avec une politique *plus court chemin*, afin de ne pas injustement pénaliser cet algorithme en partant d'une politique aléatoire. La première ligne de chaque tableau présente le nombre d'états du MDP, la deuxième le nombre d'itérations nécessaires à l'obtention de la solution, et les trois suivantes concernent le

temps de calcul. On présente d'abord le temps d'agrégation, c'est-à-dire le temps nécessaire à construire le MDP abstrait en recalculant les fonctions de transition. Vient ensuite le temps de calcul proprement dit, c'est-à-dire le temps nécessaire à l'algorithme *Policy Iteration* pour converger. Enfin, la dernière ligne fait le total.

Tout d'abord et sans surprise, les temps nécessaires à la phase d'agrégation sont plus ou moins importants suivant la méthode utilisée, puisque celles-ci génèrent plus ou moins d'états abstraits par couloir. Ensuite, la seconde méthode est légèrement plus lente que la première, ce qui est également dû au nombre d'états à traiter. En revanche, un résultat est plus étonnant : la troisième méthode est quasiment aussi rapide que la première. En effet, le temps d'agrégation est plus long, mais le temps de résolution du MDP est plus court, ce qui donne un temps total quasiment aussi bon, voire meilleur dans le cas du second environnement. En fait, les seconde et troisième méthodes nécessitent moins d'itérations que la première (sans que nous n'y ayons trouvé une explication), ce qui accélère les temps de calcul.

Mais le résultat le plus important de ce tableau est le gain en temps de calcul que nous constatons. En effet, par rapport au calcul de la solution optimale sur l'environnement de base, les temps sont divisés respectivement par 9,8, 11,2 et 5,8 sur nos trois environnements. C'est sur le deuxième environnement que les gains sont les plus intéressants, puisque c'est sur celui-ci que l'espace d'états est le plus réduit.

	Méthode 1	Méthode 2	Combinaison des 2	Solution optimale
Nombre d'états	646	916	802	3753
Nombre d'itérations	8,22	7,36	7,47	9,65
Temps d'agrégation	26	46	37	0
Temps de calcul	128	120	120	1510
Temps total	154	166	156	1510

TAB. 4.5 – *Temps de calcul pour l'environnement de la figure 4.19*

	Méthode 1	Méthode 2	Combinaison des 2	Solution optimale
Nombre d'états	280	404	320	2460
Nombre d'itérations	7,51	6,53	6,70	7,54
Temps d'agrégation	16	29	20	0
Temps de calcul	43	36	37	631
Temps total	59	65	57	631

TAB. 4.6 – *Temps de calcul pour l'environnement de la figure 4.20*

4.7.4 Robustesse aux obstacles

Comme nous l'avons indiqué précédemment, les résultats présentés sur les environnements des figures 4.19 et 4.20 sont des résultats moyens sur 10 instances de chaque

	Méthode 1	Méthode 2	Combinaison des 2	Solution optimale
Nombre d'états	436	632	594	1640
Nombre d'itérations	6,38	6,15	5,46	7,38
Temps d'agrégation	2,4	4,6	3,9	0
Temps de calcul	20,4	21,9	19,1	134
Temps total	22,8	26,5	23	134

TAB. 4.7 – *Temps de calcul pour l'environnement de la figure 4.21*

environnement. La première instance contient un grand nombre d'obstacles, la dernière n'en contient aucun. Sur les figures qui suivent, l'environnement numéro 1 (à gauche) est le plus obstrué, alors que le numéro 10 (à droite) ne contient aucun obstacle. Après avoir donné en détail les résultats obtenus, nous examinons successivement la robustesse aux obstacles de nos trois méthodes.

Résultats détaillés

Les tables 4.8 et 4.9 présentent les résultats obtenus sur les dix instances de nos deux environnements de test. La colonne la plus à gauche correspond à l'environnement comprenant tous les obstacles, et la colonne la plus à droite donne les résultats de l'instance ne comportant aucun obstacle.

Méthode 1	80,33	79,68	77,43	75,65	71,91	65,91	62,19	58,90	57,79	54,61
Méthode 2	88,70	87,81	88,04	89,70	89,41	89,80	88,42	90,87	91,06	95,59
Combinaison	88,97	88,12	88,36	89,48	89,26	90,28	88,88	91,43	91,48	95,88

TAB. 4.8 – *Détail des résultats sur les 10 instances de la Figure 4.19.*

Méthode 1	91,88	89,45	89,21	88,10	84,75	82,82	85,08	78,90	70,38	67,06
Méthode 2	94,13	94,70	94,73	94,15	93,69	94,31	94,27	94,94	96,88	96,91
Combinaison	94,39	94,98	95,06	94,70	93,88	94,57	94,96	95,44	97,15	97,86

TAB. 4.9 – *Détail des résultats sur les 10 instances de la Figure 4.20.*

On peut noter brièvement que dans chaque cas la première méthode est moins bonne que les autres, et que la combinaison des deux méthodes est généralement meilleure que l'utilisation de la seconde méthode seule. Deux exceptions tout de même : les instances 4 et 5 du premier environnement, pour lesquelles la deuxième méthode est très légèrement meilleure.

Première méthode

La première méthode d'agrégation a un comportement très similaire à l'algorithme *plus court chemin*, comme on peut le voir sur les figures 4.22 et 4.23. En effet, ces deux

algorithmes ont en commun de ne pas tenir compte de la largeur des couloirs, ce qui amène une dégradation des résultats au fur et à mesure que les obstacles sont retirés de l'environnement. Quand il y a beaucoup d'obstacles dans un couloir, il n'y a pas beaucoup de façons différentes de le traverser : il faut éviter les obstacles, donc les politiques calculées par de tels algorithmes ne pourront pas s'éloigner beaucoup de la politique optimale. En revanche, s'il n'y a pas d'obstacle, une politique stochastique va privilégier les états au milieu du couloir (minimisant ainsi les risques de collision), ce que ne fait pas une politique *plus court chemin*. Ceci explique la baisse régulière de qualité.

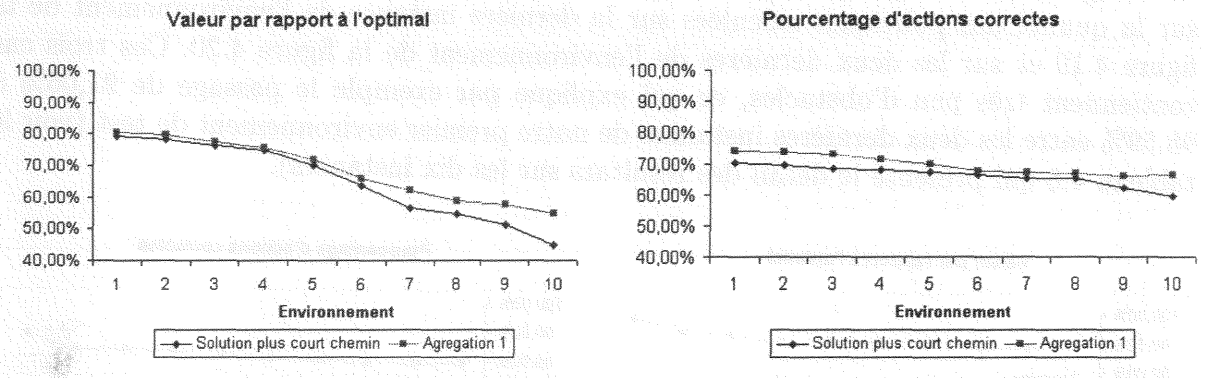


FIG. 4.22 – *Méthode 1 : qualité sur les 10 instances de la Figure 4.19*

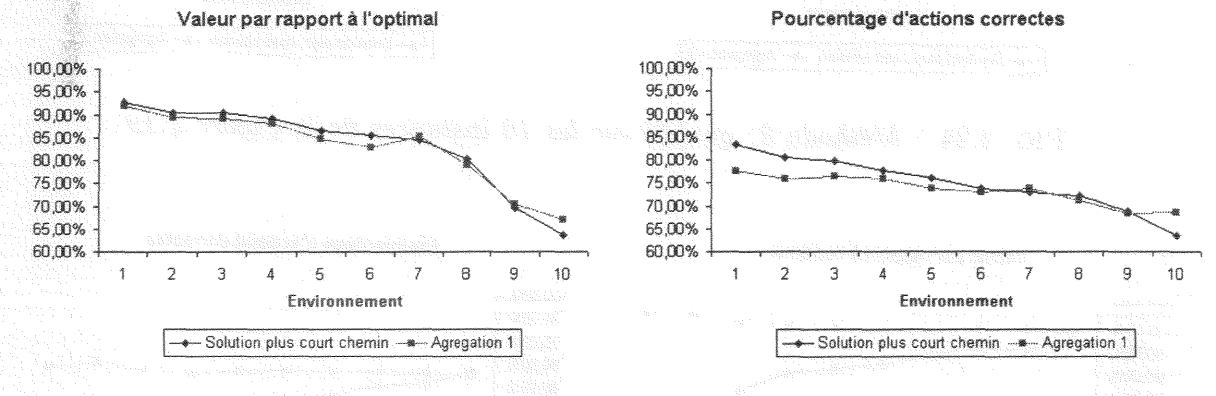


FIG. 4.23 – *Méthode 1 : qualité sur les 10 instances de la Figure 4.20*

Cette méthode d'agrégation est donc inefficace dans les environnements comportant des couloirs larges, quel que soit le niveau d'obstruction des couloirs. Elle est donc à réserver uniquement à des environnements dont les couloirs sont étroits (largeur inférieure ou égale à deux).

Deuxième méthode

La deuxième méthode, qui comme nous l'avons vu précédemment, donne de biens meilleurs résultats que la précédente, est également légèrement moins sensible à la présence d'obstacles, comme on peut le voir sur les figures 4.24 et 4.25. Elle est sur chaque instance

meilleure que l'algorithme *plus court chemin*, et, contrairement à la première méthode, a même tendance à s'améliorer au fur et à mesure que l'on enlève des obstacles. Ceci a une explication simple : quand une erreur est faite, elle est faite pour tout un couloir, puisque une seule action optimale est affectée à chaque couloir. Cette erreur va entraîner le choix d'un chemin sous-optimal, erreur qui sera plus ou moins grave selon la présence d'obstacles. Si le chemin sous-optimal contient des obstacles, le risque de collision augmente, ce qui pénalise fortement la qualité de la politique. En revanche, s'il n'y a pas d'obstacles, le fait de choisir un chemin sous-optimal va uniquement allonger le chemin à parcourir, mais sans augmenter le risque de collision. Ceci explique également le « saut » qu'on observe sur la qualité des politiques calculées sur la dernière instance de l'environnement de la figure 4.19 et sur les deux dernières de l'environnement de la figure 4.20. Ces trois cas contiennent très peu d'obstacles, ce qui explique par exemple le passage de 91,06% à 95,59% entre les deux dernières instances de notre premier environnement de test (voir le tableau 4.8 qui présente le détail des résultats sur les dix instances).

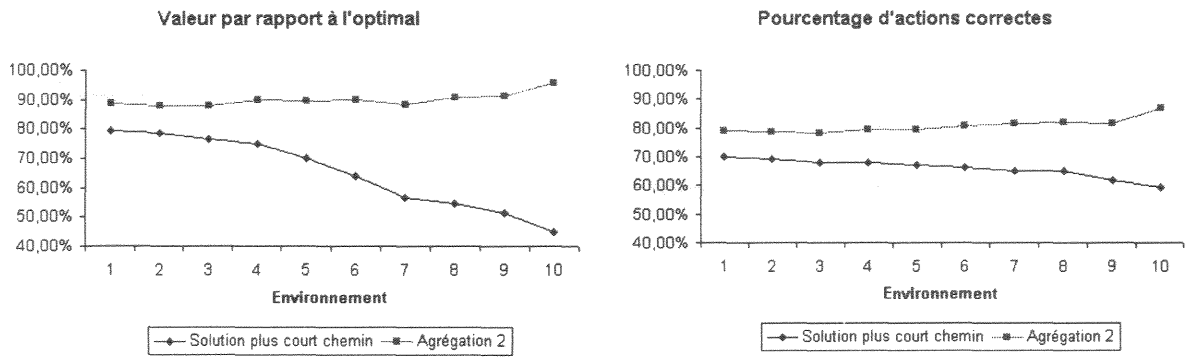


FIG. 4.24 – *Méthode 2 : qualité sur les 10 instances de la Figure 4.19*

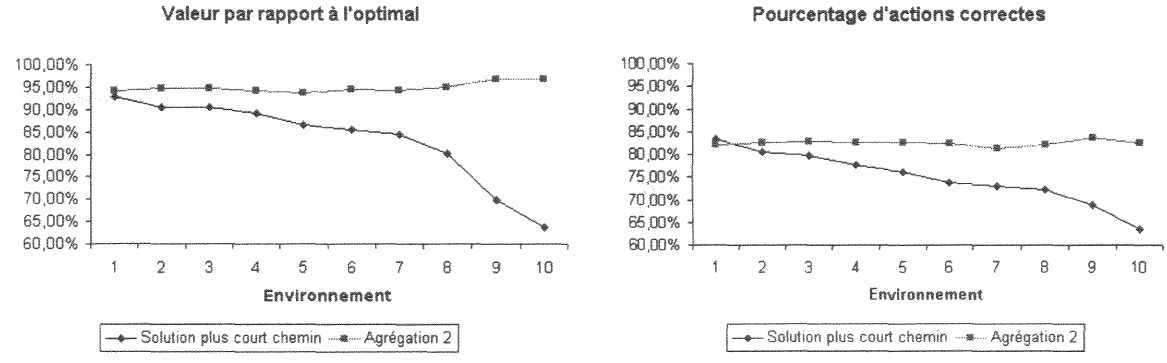


FIG. 4.25 – *Méthode 2 : qualité sur les 10 instances de la Figure 4.20*

Cette seconde méthode est donc plus robuste à la présence d'obstacles, mais la différence vis-à-vis d'un algorithme *plus court chemin* (beaucoup plus rapide) n'est pas très importante sur de petits environnements qui comportent beaucoup d'obstacles. En effet, si on gagne 9,4% sur la première instance du premier environnement de test, le gain n'est que

de 1,38% sur le second. En revanche, dès qu'on retire des obstacles, la différence s'accentue considérablement et notre méthode devient alors de plus en plus intéressante vis-à-vis de l'algorithme *plus court chemin*: sur la dernière instance des deux environnements, le gain est respectivement de 50,79% et 33,31%.

Combinaison des deux

Les résultats obtenus par la combinaison des deux méthodes ont globalement le même comportement que ceux donnés par la seconde. Les courbes représentées figures 4.26 et 4.27 sont en effet sensiblement identiques aux courbes précédentes. Des remarques identiques s'appliquent donc à cette méthode.

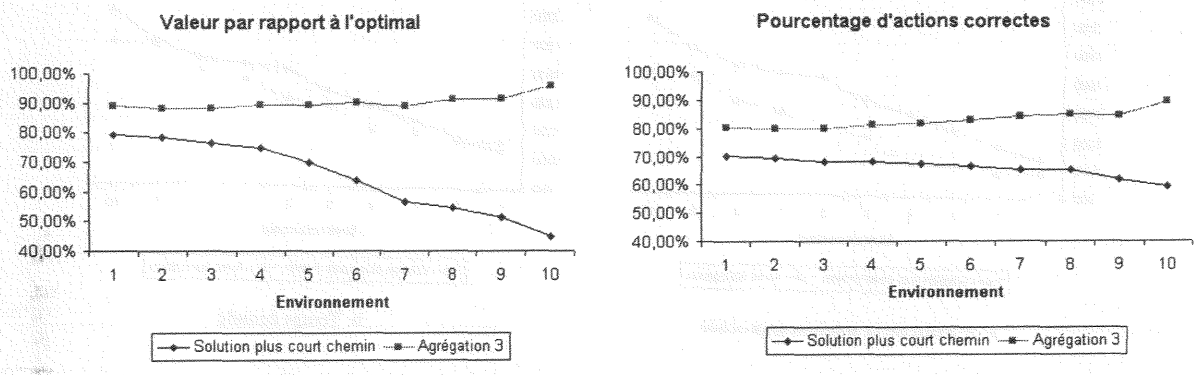


FIG. 4.26 – *Méthode 3: qualité sur les 10 instances de la Figure 4.19*

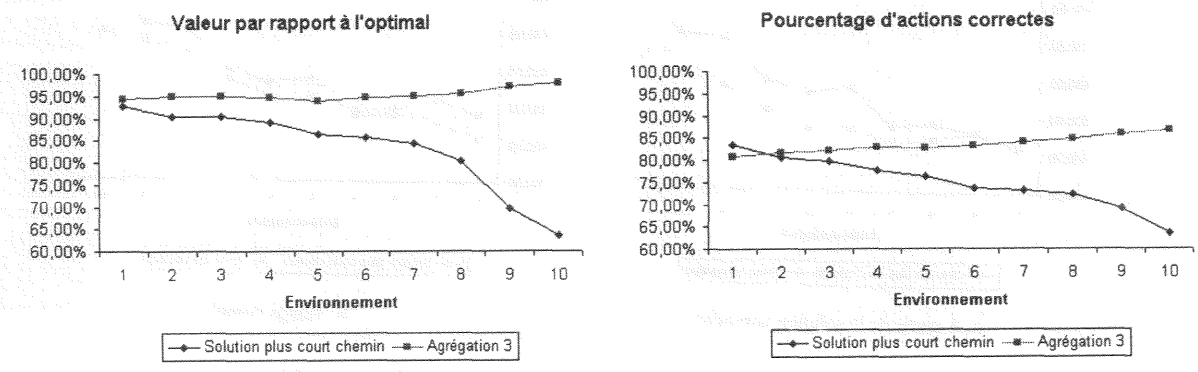
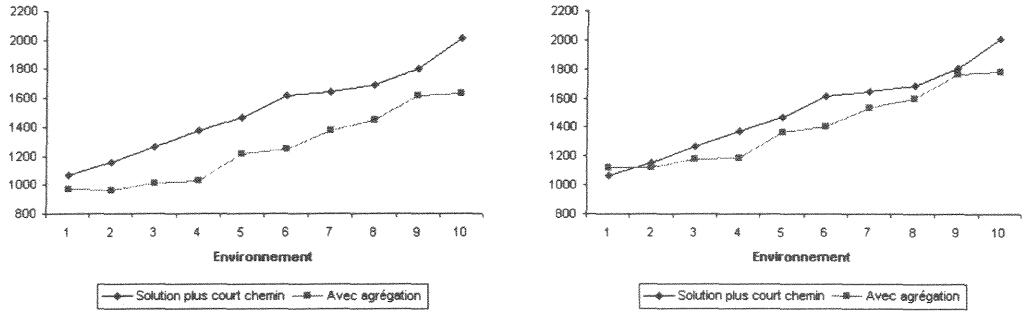


FIG. 4.27 – *Méthode 3: qualité sur les 10 instances de la Figure 4.20*

4.7.5 Calcul des solutions optimales

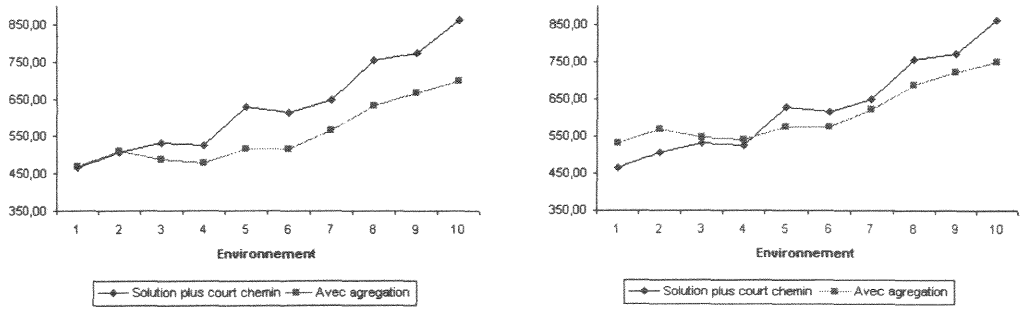
Comme nous l'avons vu dans le paragraphe 2.5.3 (page 24), *Policy Iteration* est un algorithme *anytime* et itératif: il part d'une solution quelconque qui est améliorée successivement jusqu'à obtenir une politique optimale. Notre approche permettant d'obtenir des politiques proches de l'optimal, nous pensions pouvoir les utiliser en entrée de *Policy*

Iteration pour obtenir très rapidement des politiques optimales. Les résultats obtenus ont été plutôt décevants. Les figures 4.28 et 4.29 comparent sur les deux premiers environnements de test les temps de calcul de *Policy Iteration* dans deux cas : en partant d'une politique *plus court chemin* et en partant d'une politique calculée avec notre approche (en utilisant la combinaison des deux méthodes). On retrouve sur ces figures les 10 instances d'environnements, le plus obstrué à gauche, celui sans aucun obstacle à l'extrême droite. Chaque figure comporte deux courbes : toutes deux indiquent le temps de calcul nécessaire pour passer d'une solution approchée à la solution optimale, mais celle de gauche ne tient pas compte du temps nécessaire pour obtenir la solution approchée, alors que ce temps est cumulé sur la figure de droite.



a. A partir de la solution approchée b. Temps cumulé

FIG. 4.28 – *Temps de calcul pour la figure 4.19.*



a. A partir de la solution approchée b. Temps cumulé

FIG. 4.29 – *Temps de calcul pour la figure 4.20.*

Quel que soit l'environnement, le gain en temps de calcul reste très faible. En ajoutant le temps nécessaire pour calculer la solution sur le modèle agrégé, les temps de calcul peuvent même être moins bons que ceux de la solution optimale. C'est le cas pour la première instance du premier environnement et pour les quatre premières du second : ici les politiques *plus court chemin* sont relativement proches des politiques approchées (en terme de qualité) et de ce fait les temps de calcul nécessités par le calcul de la solution optimale sont également très proches. En revanche, dans les environnements peu obstrués, pour lesquels les politiques *plus court chemin* sont très peu efficaces, les gains en

temps de calcul sont plus intéressants : 226 secondes sur la dernière instance du premier environnement et 111 secondes pour la dernière instance du second. Globalement, le temps de calcul moyen passe de 1510 à 1405 secondes (156 secondes nécessaires à l'obtention de la solution du modèle agrégé et 1249 secondes pour converger vers la solution optimale) sur le premier environnement, et de 631 à 612 secondes (57 plus 555) sur le second. Les gains en temps de calcul sont donc très faibles, ce qui correspond au nombre d'itérations nécessaires à la convergence vers la solution optimale : de 9,65 à 8,1 pour le premier environnement, et de 7,54 à 7,5 sur le second. Bien que partant de solutions bien meilleures, l'algorithme nécessite encore beaucoup d'itérations avant la convergence. Ceci est dû au fonctionnement de *Policy Iteration* : les premières itérations permettent de beaucoup améliorer la politique initiale, alors que les dernières passent beaucoup de temps à affiner des détails qui ont peu d'influence sur la valeur globale de la politique.

4.8 Profitons du navigateur !

4.8.1 Introduction

Dans les paragraphes précédents, nous avons proposé trois méthodes d'agrégation d'états et présenté leurs résultats. Pour obtenir une bonne politique, il s'agit principalement :

1. D'éviter les obstacles ;
2. De placer le robot au milieu des couloirs ;
3. D'emprunter les chemins qui présentent le meilleur compromis entre la distance à parcourir et les obstacles qui les parsèment ;
4. De découper les couloirs de façon assez fine pour que l'affectation d'une seule action par couloir ait un sens. En effet, si l'affectation d'une unique action à un couloir de deux mètres de long peut être très efficace, ce n'est plus forcément vrai si ce couloir a une longueur de 20 mètres. Dans certaines parties de ce couloir, il sera optimal de joindre une intersection, alors que depuis une autre partie du couloir, il faudra atteindre une autre intersection.

Ces quatre critères remplis, on peut raisonnablement espérer obtenir des politiques très proches des politiques optimales. Mais toutes les caractéristiques d'une « bonne » politique sont-elles indispensables à une bonne exécution des missions ? Est-il possible qu'une « mauvaise politique » soit malgré tout aussi efficace qu'une politique optimale ? C'est ce que nous montrons en introduisant une quatrième méthode d'agrégation d'états, qui sous certaines conditions, peut s'avérer au moins aussi efficace qu'une politique optimale.

4.8.2 Exposé de la méthode

Afin d'illustrer cette méthode d'agrégation, nous utilisons l'environnement de la figure 4.30 dans lequel le but à atteindre est matérialisé par une croix, en haut à droite. La politique optimale calculée pour cet environnement est présentée figure 4.31. Sur cette figure, on retrouve les caractéristiques habituelles de nos politiques optimales : évitement des obstacles et placement au milieu des couloirs pour plus de sécurité.

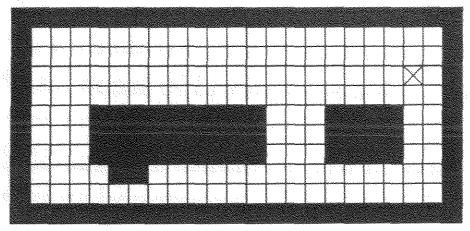


FIG. 4.30 – *Un environnement simple.*

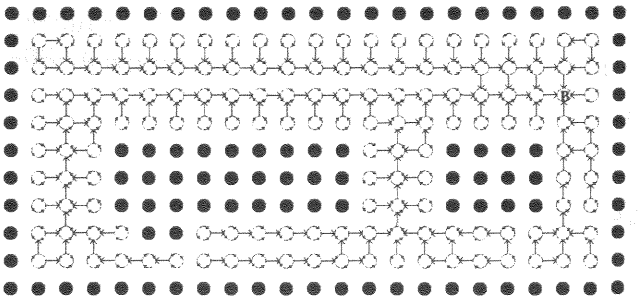


FIG. 4.31 – *Politique optimale de l'environnement de la figure 4.30.*

La figure 4.32 montre une exécution de cette politique : partant collé au mur du haut, le robot pivote immédiatement pour aller se placer au milieu du couloir, puis continue son chemin jusqu'à ce qu'il ait atteint le but. Ici la mission ne présentait aucune difficulté. En revanche, sur la figure 4.33, la même politique est exécutée dans le même environnement, mais nous y avons ajouté un obstacle qui était inconnu au moment du calcul de la politique optimale. On voit sur cette figure que le module de navigation utilisé pour superviser l'exécution de la politique (voir le paragraphe 3.5.1, page 55) évite l'obstacle par le haut, puis l'exécution reprend son cours normal. Le robot pivote pour se rapprocher du milieu du couloir, puis continue son chemin jusqu'au but. Après la manœuvre de pivotement, on remarque une autre influence du navigateur : du fait de son orientation initiale, le robot a tendance à se rapprocher du mur du bas, mais quand la distance au mur descend sous un seuil, le robot pivote alors très légèrement pour s'éloigner du mur.

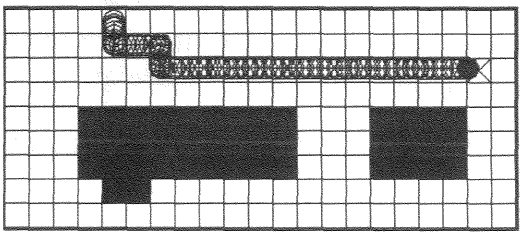


FIG. 4.32 – *Exécution de la politique optimale de la figure 4.31.*

Ces deux comportements, l'évitement d'obstacles et le positionnement au milieu du couloir, sont donc à la fois gérés par la politique à exécuter et par le module de navigation. La méthode d'agrégation présentée ici exploite ce fait : ce qui est géré par le navigateur n'a pas à être prévu par la politique. De ce fait, la tâche consistant à sortir d'un couloir

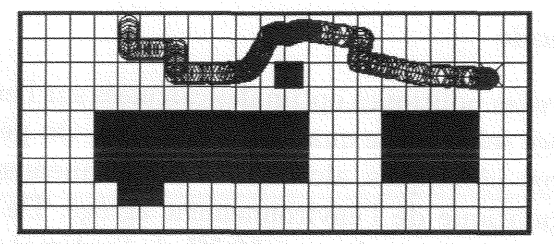
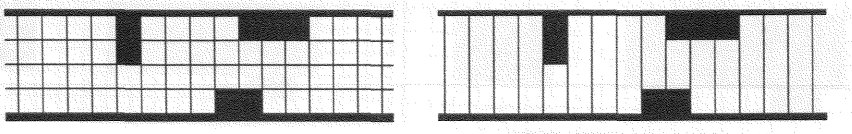


FIG. 4.33 – *Autre exécution de la même politique optimale : ajout d'un obstacle imprévu.*

est simplifiée : il ne s'agit plus d'en sortir tout en évitant les obstacles et en restant éloigné des murs, il s'agit juste d'en sortir, mais par le « bon » chemin. Ce « bon » chemin étant celui préconisé par la politique optimale. Cette méthode d'agrégation peut être résumée en trois points :

1. La largeur des couloirs n'a pas d'importance : que le couloir soit très large ou très étroit, le module de navigation est capable de positionner le robot suffisamment loin des murs pour que la mission se déroule en toute sécurité ;
2. L'évitement d'obstacles, *dans la plupart des cas*, peut être laissé à la charge du navigateur ;
3. La longueur des couloirs est au contraire très importante : dans un long couloir, la direction à prendre sera fonction de la position initiale du robot. Par exemple, dans le couloir horizontal du bas de la figure 4.31, l'intersection à atteindre dépend de la position dans laquelle le robot se trouve dans ce couloir.

Ce qui nous amène à une nouvelle façon d'agréger les états, qui est illustrée par la figure 4.34. Le couloir initial comportant 212 états de base donne lieu à la création de 30 états abstraits. Le couloir est en effet découpé en longueur, puisque la largeur n'est plus importante. Pour la même raison, seules deux orientations sont conservées, comme pour la première méthode d'agrégation expliquée précédemment. Dans un couloir horizontal, seules les orientations Est et Ouest sont conservées, et dans un couloir vertical on ne conserve que les orientations Nord et Sud. Les 30 états abstraits correspondent aux 15 positions possibles selon 2 orientations.



a. Sans agrégation : 212 états. **b. Après agrégation : 30 états.**

FIG. 4.34 – *Quatrième méthode d'agrégation.*

La fonction de transition est identique à celle générée pour la première méthode : deux actions `Traverser_Couloir` et `Demi_Tour` sont ajoutées au modèle initial. La fonction de gain n'est pas modifiée.

Intérêt de cette méthode

La figure 4.35 compare les politiques optimales calculées sur l'environnement de base et sur l'environnement agrégé selon notre quatrième méthode. On peut voir que globalement les chemins empruntés sont identiques, alors que le positionnement au milieu des couloirs n'est pas prévu par la politique de l'environnement agrégé. De ce fait, la valeur de la politique sous-optimale représente seulement 80,5% de l'optimal, ce qui est très faible.

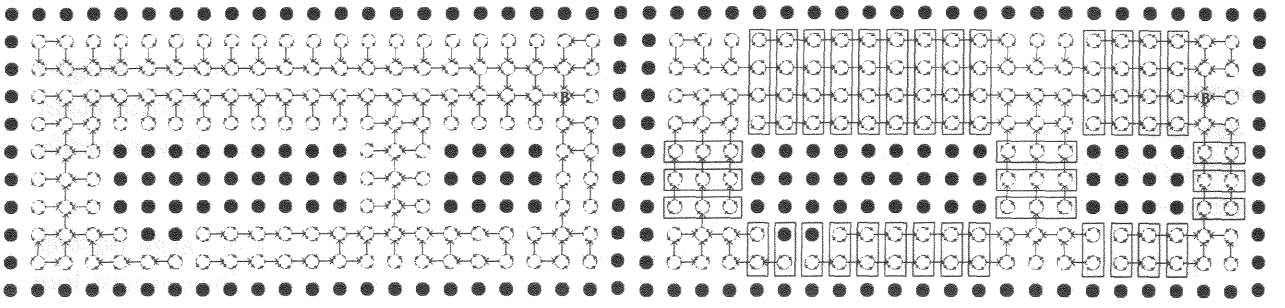


FIG. 4.35 – *Politiques optimales (1) environnement de base (2) environnement agrégé.*

Malgré ce mauvais résultat, cette politique est tout à fait acceptable, comme on peut le voir sur les figures 4.36 et 4.37. Ces figures comparent l'exécution de la politique optimale (à gauche) à l'exécution de la politique de l'environnement agrégé (à droite), sur notre environnement d'exemple. Sur la première de ces figures, on remarque que le comportement du robot est sensiblement différent selon la politique suivie. En exécutant la politique sous-optimale, le robot s'éloigne régulièrement du mur auquel il est accolé pour progressivement rejoindre le milieu du couloir, et enfin atteindre le but. Le comportement est ici beaucoup plus naturel et moins saccadé que le comportement induit par la politique optimale (puisque les actions de pivotement demandent un arrêt du robot). Sur la seconde figure, on voit que le robot est également capable d'éviter l'obstacle imprévu. La trajectoire du robot est ici aussi beaucoup plus « fluide ».

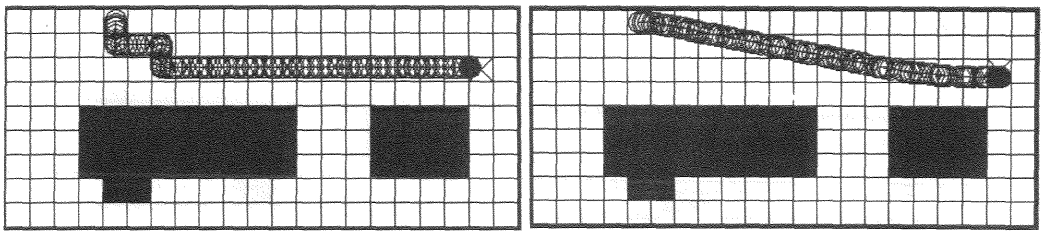


FIG. 4.36 – *Exécution des politiques (1) environnement de base (2) environnement agrégé.*

Ces deux petits exemples semblent montrer que bien que l'évaluation habituelle des politiques obtenues (en terme de valeur uniquement) indique un mauvais résultat, les politiques obtenues sont au moins aussi efficaces que les politiques optimales. Nous donnons les résultats obtenus à plus grande échelle au paragraphe suivant.

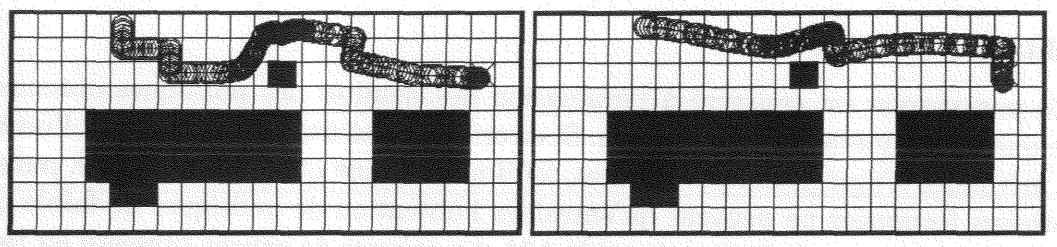


FIG. 4.37 – *Autre exécution des politiques : ajout d'un obstacle.*

4.8.3 Résultats

Dans ce paragraphe, nous donnons les résultats obtenus sur un environnement de test, présenté figure 4.38 (qui est en fait une instance peu obstruée de l'environnement de la figure 4.19). Le but a été successivement placé en vingt positions différentes. Nous présentons dans un premier temps les temps de calcul des politiques, puis les résultats en terme de qualité, puis les résultats obtenus sur l'exécution de 500 missions.

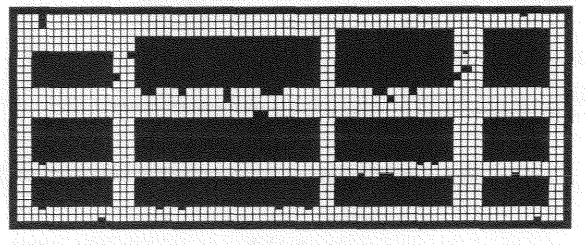


FIG. 4.38 – *Environnement de test (3976 états).*

Temps de calcul

La table 4.10 compare les temps de calcul des politiques optimales calculées sur l'environnement de base (sans agrégation d'états) aux temps nécessaires pour calculer une solution approchée en utilisant notre méthode d'agrégation. Le nombre moyen d'états des MDP est également présenté. On constate que le gain en temps de calcul correspond grossièrement au gain en nombre d'états : si l'espace d'états est divisé par 3,2, le temps de calcul est quant à lui quasiment divisé par 3. Le gain est ici peu important (comparé aux autres méthodes), ce qui n'est pas étonnant puisque nous avons choisi dès le départ d'opter pour un découpage fin des couloirs.

	Sans agrégation	Notre méthode
Nombre d'états	3976	1238
Temps de calcul	1690	570

TAB. 4.10 – *Temps de calcul en secondes.*

Qualité des politiques

Ce paragraphe est consacré à l'évaluation des qualités des politiques obtenues avec notre quatrième méthode d'agrégation. Si ce critère a été utilisé pour évaluer l'intérêt des trois méthodes d'agrégation précédentes, ici il n'est donné qu'à titre indicatif, puisque les hypothèses formulées ne peuvent qu'entraîner l'obtention de politiques de faible qualité. Les résultats sont présentés dans la table 4.11. La colonne de gauche présente les résultats obtenus en appliquant notre méthode, c'est-à-dire sans se soucier des obstacles et du positionnement au centre du couloir. La seconde colonne est donnée à titre indicatif : ces résultats correspondent aux politiques de la première colonne auxquelles on a appliqué un algorithme d'évitement d'obstacle. La troisième colonne, comme précédemment, présente les résultats obtenus par un algorithme *plus court chemin.*

	Notre méthode	Idem sans collision	Solution *plus court chemin*
Qualité moyenne	52,58	59,48	54,77
Qualité minimum	40,21	46,79	38,12
Qualité maximum	68,16	73,46	70,36
% Actions moyen	67,81	69,15	65,26
% Actions minimum	65,67	66,93	58,58
% Actions maximum	70,22	71,60	73,84

TAB. 4.11 – *Résultats pour notre environnement de test.*

Comme prévu, les résultats sont très faibles, puisqu'ils sont quasiment équivalents à ceux des politiques *plus court chemin.* Le paragraphe suivant permet d'évaluer la qualité réelle de cette dernière méthode d'agrégation.

Exécution des politiques

Dans ce paragraphe, nous présentons les résultats *réels* de cette quatrième méthode d'agrégation. Puisque certains comportements sont pris en charge par le module de navigation, la qualité des politiques générées ne peut être jugée qu'à la lumière de leur exécution. Nous avons donc testé notre approche sur 500 missions. Le but est successivement placé en 20 positions différentes, et pour chaque but à atteindre, le robot a exécuté 25 missions (5 exécutions partant de 5 positions initiales différentes). Ces 500 missions ont été exécutées en utilisant les politiques optimales, puis en exécutant les politiques obtenues avec notre méthode d'agrégation. La table 4.12 présente les résultats obtenus selon trois facteurs :

- Le temps moyen nécessaire à remplir une mission (en secondes) ;
- Le nombre moyen d'actions exécutées pour atteindre le but.
- Le nombre de missions exécutées avec succès : le robot est arrivé à moins d'un mètre du but et n'a pas touché d'obstacles sur son chemin ;

Les deux premiers chiffres ne sont pas surprenants. En effet, selon la politique sous-optimale, il n'est pas possible de pivoter dans un couloir. Placé contre un mur, le robot va s'en éloigner progressivement au lieu de pivoter. Il faut donc exécuter moins d'actions

	Politique optimale	Politique avec agrégation
Nombre d'actions moyen	107	80
Temps moyen	311	244
Nombre de succès	390	420

TAB. 4.12 – *Exécution des politiques.*

qu'en suivant une politique optimale. De plus, puisque le robot est arrêté avant chaque action de pivotement, le temps d'exécution est forcément pénalisé. En revanche, le troisième chiffre donné dans ce tableau, à savoir le nombre de missions exécutées avec succès, est lui plus inattendu. En effet, ce chiffre montre que nos politiques sous-optimales sont en fait plus efficaces que les politiques optimales. Pour comprendre ce résultat, il faut se rappeler le fonctionnement de notre architecture. Après chaque action, l'ensemble d'états dans lequel le robot peut se trouver est remis à jour, en utilisant les perceptions faites et la fonction de transition. A chaque pivotement, la fonction de transition prévoit une probabilité que celui-ci ne soit pas exécuté correctement, entraînant ainsi une incertitude non sur la position mais sur l'orientation du robot. De ce fait, après plusieurs pivots, il peut arriver par exemple que le robot, en fait orienté au nord, soit pour notre architecture orienté à l'est. De telles erreurs sont beaucoup plus graves que les erreurs de position. En effet, si une erreur de ce type intervient à l'intersection d'un couloir vertical et d'un couloir horizontal, le robot va emprunter le mauvais couloir. Si les deux couloirs sont quasi-identiques, les perceptions du robot ne vont absolument pas l'aider à corriger son erreur. Celle-ci pourra éventuellement être corrigée en fin de couloir (par exemple, le robot peut se retrouver face au mur du fond du couloir, alors que le couloir qu'il était sensé emprunter est beaucoup plus long, ce qui permet de détecter l'erreur), mais même si elle est détectée, elle est difficile à rattraper. Comme les politiques sous-optimales ne permettent pas les pivots à l'intérieur des couloirs, les erreurs d'orientation sont moindres et les missions sont exécutées avec plus de succès.

4.8.4 Limitations de la méthode

Un bémol est toutefois à apporter ici. L'environnement dans lequel notre robot a été placé est peu pourvu d'obstacles, ce qui est un avantage pour notre méthode. Mais certaines configurations font échouer à tous les coups les missions dévolues au robot. Par exemple, si on ajoute un obstacle en 'U' à notre environnement, celui-ci ne pourra pas être évité. Les politiques optimales de l'environnement de base et de l'environnement agrégé sont présentées figure 4.39. Notre méthode étant fondée sur le fait que le module de navigation est capable d'éviter les obstacles, elle ne préconise aucune action afin de les éviter, comme on peut le constater sur la figure de droite. La figure 4.40 montre comment la politique optimale de l'environnement sans agrégation permet d'éviter l'obstacle. En revanche, la figure 4.41 montre deux tentatives d'exécution de la politique avec agrégation, toutes deux conclues par un échec. Sur la figure de gauche, le robot détecte immédiatement l'obstacle et essaie de se décaler pour l'éviter. Mais étant placé trop près de l'obstacle, le navigateur échoue dans sa tentative d'évitement. Sur la figure de droite,

le robot est initialement placé plus loin de l'obstacle : il commence par s'éloigner du mur (conformément à sa politique), puis il se déplace en ligne droite. Situé juste en face de l'obstacle en forme de 'U', ses capteurs frontaux détectent trop tard l'obstacle, et le robot finit par toucher un des murs latéraux.

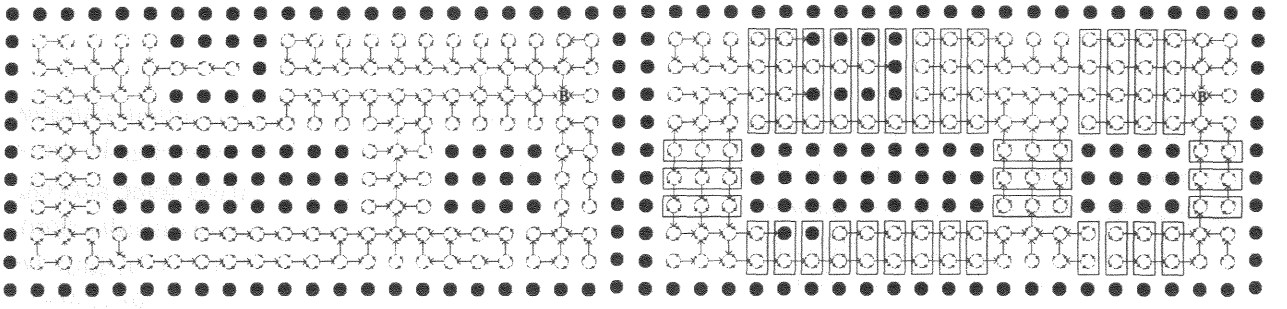


FIG. 4.39 – *Politiques optimales (1) environnement de base (2) environnement agrégé.*

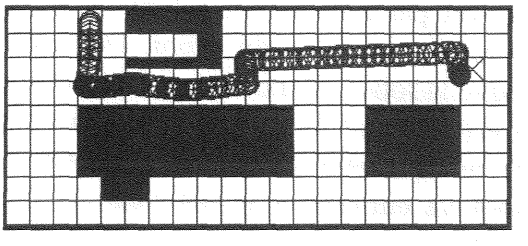


FIG. 4.40 – *Exécution de la politique optimale.*

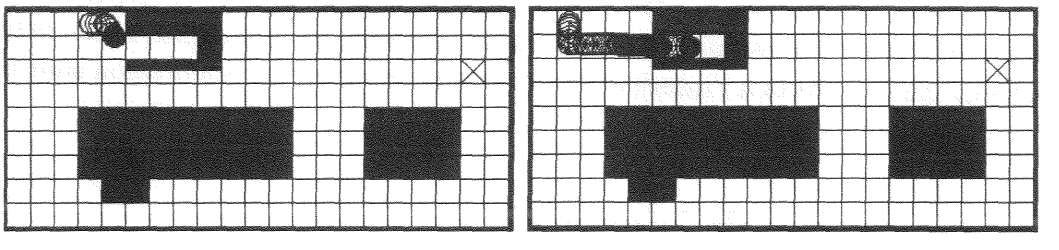


FIG. 4.41 – *Deux exécutions de la politique sous-optimale.*

Puisque cette méthode d'agrégation part de l'hypothèse que certaines tâches peuvent être laissées au module de navigation, elle trouve de ce fait ses limites dans les limites de ce module. Certains obstacles sont plus durs à éviter que d'autres, et les résultats des missions s'en ressentent. Mais l'état initial du robot a également son importance. Un obstacle normalement simple à éviter ne pourra pas être contourné si le robot est initialement placé tout contre lui. Mais ce cas de figure reste peu probable en cas réel.

4.9 Conclusion et perspectives

Dans ce chapitre, nous avons présenté plusieurs méthodes d'agrégation d'états, prenant chacune en compte de façon différente les spécificités de notre application : la planification

en environnement intérieur structuré. Le principe utilisé est simple : un couloir est fait pour être traversé de part en part. Lors de ces traversées, deux aspects sont importants :

- La façon de traverser les couloirs : en évitant les obstacles bien sûr, mais aussi en se plaçant dans une zone de sécurité, assez loin des murs pour éviter au maximum les collisions que pourraient entraîner une action mal exécutée ou une mauvaise évaluation de la position ou de l'orientation du robot ;
- Le côté du couloir par lequel le robot doit sortir. Un mauvais choix peut entraîner un détour qui retardera la bonne exécution de la mission.

Les méthodes présentées dans ce chapitre prennent en compte ces deux aspects, mais de façon différente. La première méthode, qui agrège tout un couloir en deux états abstraits, génère des politiques qui ne se soucient pas du positionnement dans la zone de sécurité au milieu des couloirs. Celles-ci sont donc de piètre qualité et l'utilisation de cette méthode doit être réservée à des environnements comportant des couloirs étroits. La seconde méthode, qui corrige ce problème en tenant compte de la largeur des couloirs, permet d'obtenir des politiques de meilleure qualité, mais celles-ci ne sont pas exemptes de défauts notamment au niveau des couloirs étroits. De ce fait, la troisième méthode s'attache à combiner les avantages des deux méthodes, et les politiques obtenues sont alors plutôt satisfaisantes, puisque nous atteignons plus de 90% de l'optimal sur notre environnement le plus complexe. Les méthodes proposées ont en plus l'avantage de très peu modifier le modèle initial puisque seule la fonction de transition est transformée, afin de prendre en compte les nouvelles actions permettant de traverser les couloirs. La fonction de gain est quant à elle inchangée et garde sa simplicité.

Mais quels sont les défauts de ces trois premières méthodes ? Tout d'abord, nous avons vu qu'elles sont peu adaptées à l'agrégation des pièces et halls. Ensuite, notre algorithme d'évitement d'obstacles n'est pas applicable à toutes situations. Les configurations en cul-de-sac, par exemple, ne peuvent être gérées par cet algorithme simple, et, fait plus grave, ce problème n'est pas non plus réglé par le module de navigation. Enfin, l'agrégation d'un couloir dans toute sa longueur n'est pas forcément la meilleure solution : si cela ne pose aucun problème lorsque l'intersection à atteindre est identique quelle que soit la position dans le couloir, cela amène à des politiques sous-optimales dans le cas général. Il serait intéressant d'étudier des découpages plus fin des couloirs, mais cela serait évidemment au détriment du temps de calcul puisque l'espace d'états serait réduit de façon moins considérable.

La quatrième méthode proposée est plus originale, puisqu'elle exploite de façon plus approfondie les spécificités de notre application. Puisque l'exécution de plans en robotique mobile nécessite l'utilisation d'un module de navigation, autant en tenir compte dès la phase de construction des politiques. Ce module est capable de placer de façon naturelle le robot au milieu des couloirs, et permet d'éviter la plupart des obstacles. Cette quatrième méthode néglige donc totalement ces deux points, afin de mieux traiter les couloirs dans leur longueur, puisque ceux-ci sont agrégés beaucoup plus finement. Le pari fait par cette méthode est d'obtenir des chemins très similaires aux chemins préconisés par la politique optimale, sans aucunement se préoccuper du comportement du robot sur ces chemins, celui-ci étant géré par le module de navigation. Nous avons montré sur un environnement que le comportement induit par les politiques générées par cette méthode

est au moins aussi efficace que celui découlant de l'exécution de politiques optimales. Cette méthode trouverait avantage à être testée sur d'autres environnements, comportant plus d'obstacles. En effet, si le module de navigation est généralement capable d'éviter les obstacles, cela dépend beaucoup de la forme de ceux-ci et du moment auquel il sont détectés. Si l'état initial du robot est situé tout contre un obstacle, celui-ci sera incapable de l'éviter. Il pourrait alors être nécessaire d'appliquer notre petit algorithme d'évitement d'obstacle, afin d'aider le navigateur dans sa tâche. Malgré tout, les configurations en cul-de-sac poseront encore des problèmes.

Ces travaux nous amènent à penser qu'il pourrait être intéressant de décrire un MDP à plusieurs niveaux de granularité, chaque zone de l'espace étant décrite plus ou moins finement, selon les besoins. Les zones fortement obstruées seraient alors uniquement constituées d'états de base, alors que les couloirs larges et comportant peu d'obstacles seraient quant à eux agrégés en un petit nombre d'états abstraits. L'idéal étant bien entendu que ceci soit fait automatiquement, en fonction de l'environnement local. C'est une des perspectives que nous voyons à ces travaux.

L'utilisation de techniques d'agrégation doit être vue comme la première étape vers une utilisation temps-réel des Processus Décisionnels de Markov en robotique. Les temps de calcul (2 minutes 30 en moyenne sur notre plus gros environnement) sont plus raisonnables, et les expériences faites pour tester la quatrième méthode semblent montrer que l'utilisation de politiques sous-optimales (mais néanmoins proches de l'optimal) reste efficace. Malgré des contraintes fortes au départ (représentation d'un couloir à l'aide de très peu d'états), l'agrégation donne de bons résultats. Cette constatation nous a motivé dans l'utilisation d'une autre méthode classique de résolution de problèmes, dont les hypothèses de départ semblent plus prometteuses : la décomposition. Le chapitre suivant expose nos travaux dans ce domaine.

Chapitre 5

Décomposition des Processus Décisionnels de Markov

5.1 Introduction

Face à un problème conséquent, il est toujours tentant d'utiliser la technique *diviser pour régner*. Prenons le cas d'une personne qui se rend de son bureau à son domicile. Elle commence par sortir de son bureau, emprunte divers couloirs jusqu'à une porte permettant de sortir du bâtiment, puis elle rejoint sa voiture, avec laquelle elle se rend à son domicile. Si le lendemain, cette personne décide de faire un crochet par la librairie de la rue du Saint-Gothard avant de rejoindre son domicile, une grande partie du plan reste valable : quel que soit l'endroit où la personne se rend une fois quitté le bâtiment, la stratégie utilisée pour en sortir est la même. Ceci est tout à fait applicable à la résolution de MDP. On parle dans ce cas de *décomposition* de MDP. *Décomposer* un MDP consiste à définir des sous-ensembles de l'espace d'états, chaque sous-ensemble constituant un MDP partiel qui est résolu indépendamment. La politique globale (solution du MDP global) est obtenue par la réunion des solutions partielles. Selon la qualité de ces solutions partielles, cette politique pourra être optimale ou sous-optimale.

Plusieurs problèmes sont alors à résoudre :

- Etant donné un environnement, comment trouver une partition optimale de l'espace d'états ? Chaque partition donnant lieu à un MDP partiel ne doit contenir qu'un nombre limité d'états, afin que la résolution de chaque MDP partiel soit rapide. Cette partition doit néanmoins être réaliste vis-à-vis du problème traité, afin que les états d'une partition partagent des caractéristiques communes sans lesquelles la partition n'aurait aucun sens.
- Comment représenter les liens entre partitions ? En effet, quand on dit que les MDP partiels sont résolus indépendamment, ce n'est pas tout à fait vrai. Prenons par exemple le problème simple représenté par les figures 5.1 et 5.2 : partant de l'intérieur d'un bureau, il s'agit d'arriver au but, situé dans l'un des deux autres bureaux. L'environnement est partitionné en quatre régions : la région R_1 contient le bureau de départ, la région R_2 contient le couloir et les régions R_3 et R_4 matérialisent les bureaux dans lequels se trouve (selon le cas) le but à atteindre. Sur la figure 5.1, une

seule porte permet de sortir du bureau de départ. Il n'y a qu'un seul moyen de sortir du bureau, la position du but n'a donc aucune influence sur la politique de la région R_1. Le MDP partiel représentant cette région peut être résolu indépendamment. En revanche, sur la figure 5.2, le bureau de départ comporte deux portes permettant d'atteindre le couloir. Dans ce cas, selon la position du but (bureau à gauche ou droite du bureau de départ), le chemin optimal empruntera l'une ou l'autre porte, comme on peut le voir sur cette figure. Ceci montre que la politique optimale d'une région n'est pas toujours indépendante des autres régions. Ici, la région R_1 a deux buts possibles, les deux portes la reliant au couloir. L'attraction définie pour chacun des buts doit prendre en compte fidèlement l'emplacement du but global, sans quoi la politique définie pour la région R_1 sera sous-optimale, entraînant la sous-optimalité de la politique globale. Une méthode de décomposition efficace doit donc permettre de fixer précisément les valeurs affectées aux sous-buts.

- Une fois chaque MDP partiel résolu, comment combiner ces solutions partielles pour obtenir une «bonne» solution du MDP global? Si la politique obtenue est sous-optimale, comment obtenir rapidement une solution qui sera optimale?

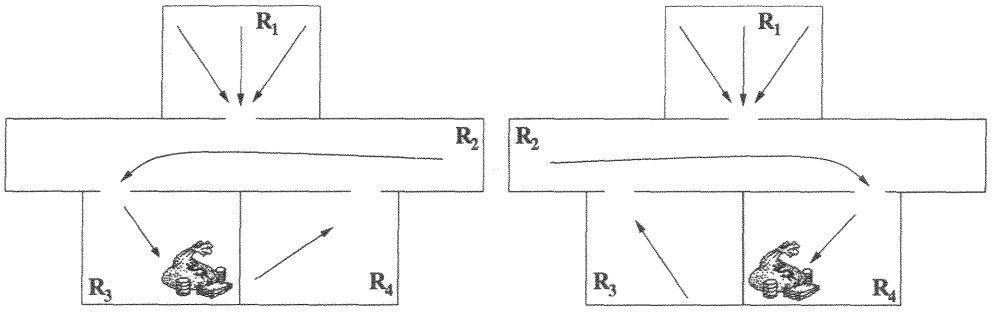


FIG. 5.1 – *Le plan de R_1 est indépendant de la position du but.*

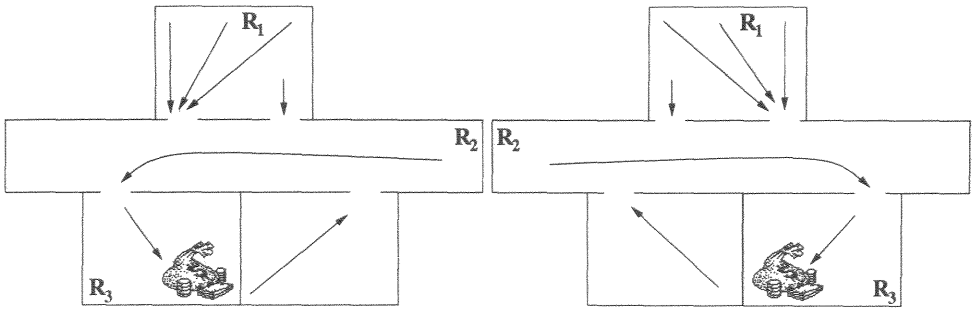


FIG. 5.2 – *Le plan de R_1 dépend de la position du but.*

Plusieurs approches ont été développées pour tenter de répondre à ces questions. Après avoir présenté les travaux les plus significatifs dans ce domaine, nous présentons notre méthode. Partant du problème de la navigation d'un robot mobile dans un environnement intérieur structuré, nous nous appuyons sur les caractéristiques de cette application pour représenter l'environnement à l'aide d'un graphe qui définit naturellement la partition de

l'espace et les liens entre ces partitions. Nous proposons successivement deux méthodes différentes de valuation de ce graphe, avant de donner l'algorithme de construction du graphe. Puis nous montrons comment le graphe est utilisé pour construire et résoudre des MDP partiels, avant de donner les résultats obtenus sur nos environnements de test et de conclure sur ces travaux.

5.2 Approches similaires

5.2.1 Travaux de *Dean* et *Lin*

Une méthode générale de décomposition de MDP est donnée dans [Dean et Lin, 1995] et [Marion, 1996]. L'espace d'états est partitionné en *régions*, chaque région donnant lieu à un MDP partiel résolu indépendamment. La solution du MDP global est obtenue en combinant ces solutions partielles. La partition de l'environnement est considérée comme une donnée du problème, fournie par un expert du domaine d'application.

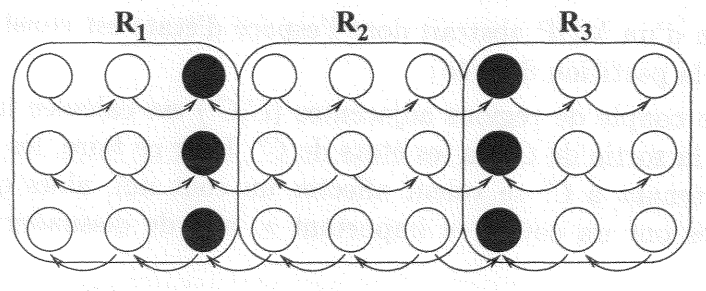


FIG. 5.3 – *Etats regroupés en régions*

La figure 5.3 montre comment les régions sont transformées en MDP partiel. Le MDP partiel correspondant à la région R_2 réunit tous les états *de base* de cette région, auxquels on ajoute ses états *périphériques* (en noir sur la figure). Ces états sont ceux qui sont immédiatement adjacents à la région (c'est-à-dire qu'on peut les atteindre en exécutant une seule action depuis un état de R_2). La périphérie d'une région C quelconque de l'environnement est définie comme suit :

$$\texttt{Peripherie}(C) = \{s' | s' \notin C \wedge \exists s \in C, a \in \mathcal{A}, T(s,a,s') > 0\}$$

Le MDP correspondant à une région quelconque C est défini de la façon suivante :

1. L'espace d'états du MDP M_C est constitué par : $C \cup \texttt{Peripherie}(C)$
2. La fonction de transition $T'(s,a,s')$ de M_C est générée comme suit :
   - $T'(s,a,s') = T(s,a,s') \quad \forall s \in C$
   - $T'(s,a,s) = 1 \quad \forall s \in \texttt{Peripherie}(C)$
3. La fonction de récompense $R(s,a,s')$ de M_C est générée comme suit :
   - $R'(s,a,s') = R(s,a,s') \quad \forall s,s' \in C$
   - $R'(s,a,s') = \lambda_{s'} + R(s,a,s') \quad \forall s \in C, \quad \forall s' \in \texttt{Peripherie}(C)$

- $R'(s,a,s) = 0 \quad \forall s \in \texttt{Peripherie}(C)$

Les états périphériques sont donc définis comme des états absorbants, afin que leur attraction soit la plus forte possible, puisque le but est de sortir de la région. L'attraction des régions périphériques vis-à-vis de C est définie par le facteur λ, qui est la clé d'une bonne décomposition. Si ce facteur est égal à la valeur donnée par la politique optimale à l'état périphérique, alors la politique locale définie sur M_C sera optimale pour le MDP global. Bien entendu, cette valeur n'est pas connue, et les auteurs donnent alors plusieurs méthodes permettant d'approcher ou de déterminer ce facteur.

Construction hiérarchique de stratégie

Le premier algorithme proposé est appelé « Construction hiérarchique de stratégie » (*Hierarchical Policy Construction*). Le MDP global est résolu en combinant des techniques d'agrégation et de décomposition. Cet algorithme résout un MDP abstrait, en suivant le plan ci-dessous :

1. Construction d'un MDP abstrait dont l'espace d'états est constitué par les régions définies par la partition donnée ;
2. Pour chaque couple de régions adjacentes (C,C') est calculée une politique $\pi_{C \to C'}$ qui favorise la sortie de C via les états de C'. Pour ce faire, les états périphériques de C appartenant à C' se voient affecter un coût nul, alors que les autres états périphériques ont un coût κ « important », afin de dissuader le passage par ces états :
$$\lambda_s = 0 \quad \forall s \in C' \cap \texttt{Peripherie}(C)$$
$$\lambda_s = \kappa \quad \forall s \in \texttt{Peripherie}(C) - C'$$
3. L'ensemble de ces actions de haut-niveau permettant de passer d'une région à l'autre constitue l'ensemble d'actions applicables dans le MDP abstrait (à partir d'un état abstrait C, il sera possible d'exécuter toutes les actions $\pi_{C \to X}$) ;
4. La fonction de transition entre états abstraits (c'est-à-dire entre régions) doit être recalculée (en utilisant les fonctions de transition initiales) pour chaque couple (état,action) ;
5. La fonction de gain doit également être recalculée, car elle prend elle aussi en compte les nouvelles actions ;
6. Enfin, la solution du MDP abstrait est calculée de façon classique. La politique obtenue spécifie pour chaque région initiale C une action $\pi_{C \to X}$, où X est la région adjacente à atteindre. Pour connaître l'action à exécuter sur un état s de base de C, il suffit d'utiliser $\pi_{C \to X}(s)$.

De l'aveu même des auteurs, cet algorithme ne permet de trouver une politique globale optimale que dans des cas particuliers (voir [Marion, 1996], page 105). En effet, les politiques générées spécifient une seule région à atteindre par région initiale. Quel que soit l'état de base de C, l'action affectée par la politique va mener à (ou se diriger vers) C'. Or en partant d'un état de base s appartenant à une région C, il peut être optimal de se diriger vers C_1, alors qu'en partant d'un autre état de base s' de C, il sera optimal

de se diriger vers C_2. En appliquant l'algorithme présenté ici, de tels comportements ne pourront être générés.

Itération de stratégie locale

Pour corriger ces imperfections, les auteurs proposent de calculer plus finement les facteurs d'attraction λ d'une région par rapport à ces régions adjacentes, en procédant de façon itérative. L'algorithme que nous présentons ici est appelé « Itération de stratégie locale » (*Iterative Approximation Approach*).

La première phase consiste en un prétraitement **indépendant du but à atteindre** du MDP global. Cette phase est assez similaire à l'algorithme précédent. Pour tout couple de régions adjacentes (C,X), un MDP est créé, qui va permettre de déterminer la valeur des coûts de passages entre C et X. Ces coûts de passage seront ensuite utilisés pour déterminer les gains λ_i affectés aux états périphériques i de C. Le calcul itératif du coût de passage de C à C' est réalisé ainsi :

- Première itération : résolution du MDP M constitué des états de C, des états de C' et des états périphériques des deux régions. Les valeurs affectées aux états périphériques sont surévaluées ($\lambda_i = \kappa$) et le MDP abstrait est résolu : on obtient une politique $\pi_{C \to C'}$.
- Seconde itération : les valeurs affectées aux états périphériques sont modifiées en fonction des valeurs données à ces états par la politique $\pi_{C \to C'}$. Le MDP est à nouveau résolu en utilisant ces nouveaux paramètres.
- Ce même processus est répété jusqu'à la convergence des valeurs de transition.

Ce processus est exécuté pour tous les couples de régions adjacentes. Cette phase permet d'obtenir pour chaque région le coût relatif de sortie par ses états périphériques, indépendamment du but global de l'environnement. Ce processus est donc considéré comme un prétraitement, dont le temps d'exécution ne doit pas être pris en compte dans le temps de calcul des politiques.

La seconde phase de l'algorithme consiste à résoudre le MDP global, en fonction du but à atteindre. Tout d'abord, un MDP abstrait est résolu (comme précédemment, son espace d'états est constitué par les régions de l'environnement). Cela permet d'obtenir une valeur $V(C)$ pour chaque région C. Puis, la solution du MDP de base est calculée, en résolvant un MDP par région de l'environnement. Ici aussi, le processus est itératif. Pour toute région C, on procède ainsi :

1. Création d'un MDP composé des états de base de C et de ses états périphériques ;
2. Les valeurs des états périphériques i de C (appartenant à X) sont définies ainsi :
   - $\lambda_i = V_{\pi_X}(i)$ si π_X a déjà été calculé ($V_{\pi_X}(i)$ est la valeur de l'état i telle que définie par la politique partielle sur la région X) ;
   - $\lambda_i = max(V(C),V(X))$ sinon ($V(C)$ est la valeur de l'état abstrait C donnée par la solution du MDP abstrait).
3. Résolution du MDP ;

Ce processus est répété jusqu'à ce que les valeurs λ_i ne puissent plus être améliorées. A ce stade, la réunion des politiques partielles π_C permet d'obtenir une solution optimale pour l'environnement global.

En conclusion, ce second algorithme a l'avantage de calculer une solution optimale, mais le processus semble complexe. Tout d'abord, il nécessite un prétraitement itératif et coûteux en temps de calcul. Lorsqu'on travaille souvent sur le même environnement, ce prétraitement ne doit être calculé qu'une seule fois, on peut donc négliger son temps de calcul. Ce n'est évidemment pas le cas si on travaille sur de nombreux environnements dans lesquels on résout peu de problèmes. Mais la seconde phase de l'algorithme est également itérative, puisque les valeurs des états périphériques sont successivement raffinées en fonction des politiques partielles successives. Globalement, cet algorithme nous semble donc très complexe et est donc, selon nous, à réserver à la résolution de problèmes nécessitant un espace d'états très important.

5.2.2 Utilisation de macro-actions

D'autres recherches ont pour but de combiner astucieusement des *macro-actions* (encore appelées *options*) pour accélérer les temps de résolution de MDP : il s'agit principalement des travaux de *Parr*, *Hauskrecht et al.* et *Precup et Sutton*. La notion de macro-action est d'ailleurs présente dans les travaux de *Dean* et *Lin*, puisqu'une macro-action est en fait une politique partielle calculée sur une région. Ces travaux ont en commun de s'appuyer sur une partition de l'environnement en régions considérée comme une donnée du problème. Sur chaque région est calculé un ensemble de politiques, qui seront autant de macro-actions applicables aux régions correspondantes. Un MDP abstrait est résolu en utilisant ces macro-actions : la politique obtenue affecte à chaque région une macro-action, c'est-à-dire une politique partielle qui permet d'atteindre les régions adjacentes. Selon les travaux, les politiques globales (obtenues par la réunion des politiques partielles de chaque région), sont soit sous-optimales soit optimales. En effet, la qualité de la politique globale dépend de la qualité des ensembles de macro-actions calculées dans chaque région. Si ces ensembles ne contiennent pas de « bonnes » politiques, la politique globale ne peut pas être optimale. Mais le calcul de politiques partielles a un coût non négligeable, on ne peut donc se permettre de calculer un trop grand nombre de macro-actions par région. Ces approches ont donc en commun d'utiliser des techniques permettant de ne calculer qu'un ensemble minimal de macro-actions par région, en s'assurant que celui-ci contienne uniquement de « bonnes » politiques.

Dans les travaux de *Parr* [Parr, 1998a; Parr, 1998b], le MDP abstrait construit est constitué uniquement des états périphériques de chaque région, ce qui permet de réduire considérablement l'espace d'états. Deux algorithmes sont proposés pour construire itérativement les ensembles de macro-actions minimaux. Le premier, intitulé *Value Space Search*, s'inspire de l'algorithme *Witness* (voir la paragraphe 2.4.3). La résolution de programmes linéaires permet de déterminer si l'ensemble de macro-actions est suffisant pour assurer une qualité ϵ-optimale en tout état périphérique ; si oui, l'ensemble de macro-actions minimal est constitué, sinon on ajoute la politique qui permet d'atteindre cette qualité. Un second algorithme appelé *Convex Hull Bounding Approach* est également proposé. Plus d'inspiration géométrie algorithmique, il utilise également des techniques de programmation linéaire pour construire l'ensemble de macro-actions. Dans les deux cas, il est possible de construire les ensembles de macro-actions pour obtenir une politique globale ϵ-optimale ou pour obtenir la politique optimale, mais dans ce dernier cas le nombre

de macro-actions à calculer par région sera plus important.

Les travaux de *Hauskrecht et al.* [Hauskrecht *et al.*, 1998] sont très similaires, mais ceux-ci proposent un autre algorithme permettant de ne calculer qu'un ensemble minimal de macro-actions. Cette approche insiste plus sur la réutilisation des macro-actions (sur des MDP identiques mais dont le but a été déplacé, ou sur des MDP similaires), et propose également d'utiliser les macro-actions pour résoudre un MDP *augmenté*. Celui-ci contient tous les états de base du MDP global, mais l'ensemble d'actions contient l'ensemble des macro-actions en plus des actions de base. Ceci permet d'assurer l'obtention de la solution optimale (puisque l'espace d'états est inchangé et que l'ensemble d'actions contient les actions de base), mais la résolution n'est pas toujours plus rapide que celle du MDP original (puisque la complexité de *Policy Iteration* est liée au nombre d'actions). En pratique, la résolution est généralement plus rapide : chaque itération est plus lente, mais l'algorithme nécessite moins d'itérations pour converger.

Ces deux approches s'inspirent en fait des travaux de *Precup* et *Sutton*, qui ont introduit la notion de macro-action (sous le nom de « option ») pour résoudre des MDP, mais en utilisant des techniques d'apprentissage par renforcement. Dans ces travaux, il s'agit de résoudre le MDP de base (l'espace d'états n'est pas réduit) mais en utilisant à la fois les actions de base et les options. Dans [Precup et Sutton, 1997], les auteurs montrent notamment comment l'utilisation d'options permet de réduire le nombre d'itérations nécessaires à la convergence de l'algorithme de résolution. Cette approche est plus détaillée dans [Precup *et al.*, 1998], qui porte plus d'attention aux transformations du modèle nécessaires à la prise en compte des options.

5.2.3 Conclusion

Les deux approches décrites ont l'avantage d'être très générales. Elles s'appliquent à toutes sortes de MDP, à condition que ceux-ci aient été préalablement partitionnés en régions. Le nombre d'états permettant de passer d'une région à l'autre doit être réduit, car sinon ces méthodes perdent de leur efficacité. Le principal reproche que nous faisons à la première approche est sa complexité : les algorithmes proposés pour calculer les paramètres de passage d'une région à l'autre sont très complexes, et il n'est pas sûr que la méthode proposée soit toujours plus efficace que les méthodes traditionnelles. L'utilisation de macro-actions présente l'inconvénient de demander beaucoup de calculs *off-line* : la qualité du plan obtenu dépend des macro-actions calculées, pour lesquelles il est nécessaire de recalculer les fonctions de gain et de transition, ce qui peut être coûteux. Par rapport aux approches citées, nous nous démarquons sur plusieurs points :

- Nous ne calculons qu'une politique par région, afin de minimiser les temps de calcul. Le but étant d'éviter que le temps passé à calculer ces politiques et à modifier les fonctions de transition et de gain fasse perdre le bénéfice de la décomposition.
- Notre approche utilise les fonctions de transition et de gain de base. Il n'est pas nécessaire de les recalculer pour prendre en compte les macro-actions, puisque nous n'utilisons pas de MDP abstrait.
- Nous prêtons beaucoup d'attention à la définition des coûts de passage d'une région à l'autre. Ces coûts sont calculés en utilisant une valuation heuristique simple et rapide

des états ; ils reflètent efficacement les comportements de la politique optimale.

5.3 Décomposition à l'aide d'un graphe valué

Le but de notre approche est de montrer que les caractéristiques de l'environnement peuvent être utilisées pour partitionner l'espace d'états et définir les liens entre régions. Pour cela, nous nous baserons sur notre application de planification d'actions en environnement intérieur structuré. Dans ce type d'environnement, il existe trois sortes d'états : les bureaux, les couloirs et les intersections. Dans la suite de ce chapitre, nous parlerons principalement de couloirs et d'intersections ; les bureaux sont considérés comme un cas particulier de couloir. Quelles sont les missions d'un robot dans un bâtiment comme le Loria ? Généralement, il s'agit d'atteindre un endroit précis de l'environnement, d'y effectuer une tâche, puis d'éventuellement revenir au point initial. Ici nous nous intéressons uniquement aux déplacements du robot. Une politique peut être vue comme une succession de couloirs à traverser, entrecoupés par des intersections dans lesquelles le robot peut éventuellement changer de direction. Lorsque le robot se trouve au milieu d'un couloir, la politique optimale le mène à une des deux intersections adjacentes. La partition de l'environnement apparaît ainsi naturellement : chaque couloir est transformé en région, dont les buts sont constitués par les intersections adjacentes. L'étape suivante consiste à affecter une récompense à chaque but, afin que l'intersection la plus « intéressante » au sens de la politique optimale soit la plus attractive. Pour fixer automatiquement ces valeurs, nous utilisons un graphe orienté et valué ; chaque intersection donne lieu à un nœud du graphe, dont les arcs sont constitués par les couloirs permettant de relier ces intersections. Nous proposons deux méthodes de valuation des arcs, que nous présentons dans les paragraphes 5.4.1 et 5.4.2. Dans le paragraphe suivant, nous montrons l'importance de ces valeurs.

5.3.1 Liens entre régions

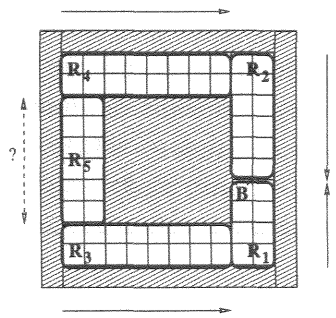


FIG. 5.4 – *Un environnement comportant 252 états.*

Prenons l'exemple simple de la figure 5.4. Le but est situé dans la partie basse du couloir vertical de droite (il est matérialisé par la lettre 'B'). L'environnement a été préalablement partitionné en 5 régions (délimitées par un trait gras et notées $R_1 ... R_5$). Les flèches de part et d'autre de l'environnement donnent une politique intuitive : par exemple,

dans les couloirs horizontaux (régions R_3 et R_4), la politique optimale consiste à traverser ces couloirs de la gauche vers la droite. Mais quelle est la politique optimale dans le couloir vertical de gauche (région R_5) ? La figure 5.5 montre deux politiques optimales calculées pour cet environnement, correspondant à deux positions distinctes du but. Si la politique est identique dans les deux couloirs horizontaux, on voit bien que celle du couloir vertical de gauche n'est pas du tout la même. Dans la figure de gauche, la plupart des états mènent vers le couloir du bas, alors que c'est le couloir du haut qui est majoritairement emprunté dans le second cas. Ceci est bien entendu dû à la position du but, qui a une influence très importante sur les politiques des MDP partiels. Ainsi, lorsqu'on construit les MDP partiels, ce facteur doit être pris en compte. Cette région de l'environnement a également une autre particularité : tous les états ne mènent pas à la même intersection. Selon la position dans le couloir, la politique optimale mène soit vers le haut, soit vers le bas. Le MDP partiel correspondant à la région R_5 devra refléter ce comportement. Intuitivement, on peut dire qu'il devra comporter deux buts : l'un au nord, l'autre au sud. Selon la position du but global, l'attraction de ces buts devra être plus ou moins forte, comme on le voit sur la figure 5.6.

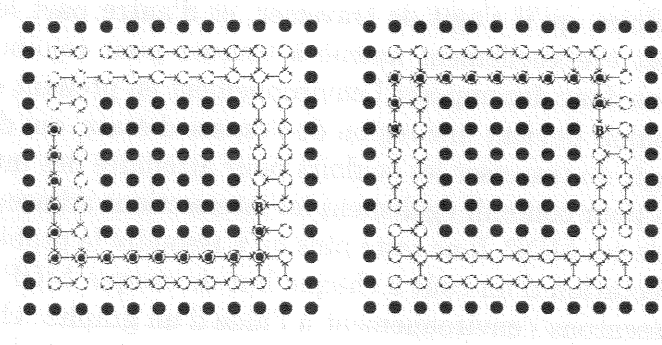


FIG. 5.5 – *2 politiques optimales pour la figure 5.4.*

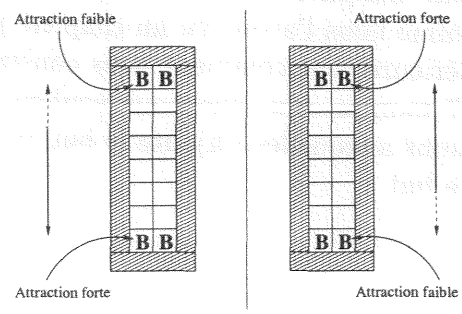


FIG. 5.6 – *Attraction des buts en fonction de la position du but global.*

Une bonne décomposition n'est donc possible que si les valeurs des buts de chaque région sont « bien » définies. Dans [Dean et Lin, 1995], les auteurs montrent le résultat théorique suivant : si la valeur des buts des MDP partiels est égale à la valeur donnée à ces mêmes états par une politique optimale sur l'environnement global, alors la combinaison des politiques locales donne une politique globale optimale. Bien évidemment, la politique globale optimale n'est pas connue (puisque c'est ce qu'on cherche !), on ne peut donc

connaître *a priori* les valeurs optimales des états buts. Dans le paragraphe suivant, nous montrons comment nous utilisons un graphe valué pour estimer ces valeurs.

5.3.2 Utilisation d'un graphe

L'idée de base de notre technique de décomposition s'appuie sur une reformulation de la politique optimale. Imaginons un robot placé dans le coin supérieur gauche de notre petit environnement (figure 5.4). Le chemin optimal le menant au but, défini par une action optimale pour chaque état constituant ce chemin, peut être reformulé ainsi :

1. Orientation du robot vers l'est ;
2. Avancer jusqu'à la prochaine intersection ;
3. Pivoter de 90° vers la droite ;
4. Avancer jusqu'au but.

Cette reformulation de la politique optimale montre bien la décomposition naturelle qui peut être effectuée dans le type d'environnement que rencontre un robot mobile. Il y a d'une part les couloirs, qu'il s'agit de traverser, et d'autre part les intersections dans lesquelles le robot est éventuellement amené à tourner pour rejoindre l'un des couloirs adjacents. Nous allons donc décomposer l'environnement en prenant en compte cette particularité : chaque couloir devient une région de l'espace d'états, qui donne lieu à un MDP partiel. Puisqu'il s'agit de traverser les couloirs pour atteindre des intersections, les MDP partiels sont étendus pour contenir également les intersections adjacentes aux couloirs, qui deviennent les buts de ces MDP. Il ne reste plus qu'à résoudre le problème principal lié à la décomposition : l'affectation d'un coût à chaque but de chaque MDP partiel. Pour définir ces coûts, nous représentons l'environnement à l'aide d'un graphe : chaque intersection de l'environnement devient un nœud, et chaque couloir (ou pièce, hall, etc.) est représenté par un arc. Ce graphe est construit automatiquement, à l'aide d'une représentation de l'environnement dans laquelle chaque état est étiqueté selon son appartenance à un couloir ou une intersection, comme nous l'avons vu au chapitre précédent (paragraphe 4.3.2, page 70). Nous verrons ultérieurement comment nous construisons le graphe, mais pour fixer les idées, la figure 5.7 montre celui que nous voulons obtenir : cinq nœuds représentent les quatre intersections auxquelles s'ajoute le but, et les arcs relient les nœuds en formant des chemins vers le but.

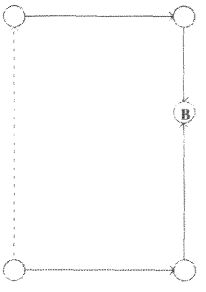


FIG. 5.7 – *Graphe représentant l'environnement de la figure 5.4.*

5.4 Valuation du graphe

Dans ce paragraphe, nous proposons deux méthodes de valuation des arcs du graphe. Par commodité, les valeurs calculées sont positives, alors que les valeurs données par une politique optimale sont toutes négatives. Quand il est fait mention de « valeur maximale » ou autres termes de ce type, il faut comprendre « valeur maximale en valeur absolue ». De plus, chaque méthode value un nœud en fonction du nœud *suivant sur le chemin menant au but.* Cet ordre dans les nœuds est obtenu dynamiquement, et nous expliciterons plus tard comment nous l'obtenons.

5.4.1 Première méthode de valuation

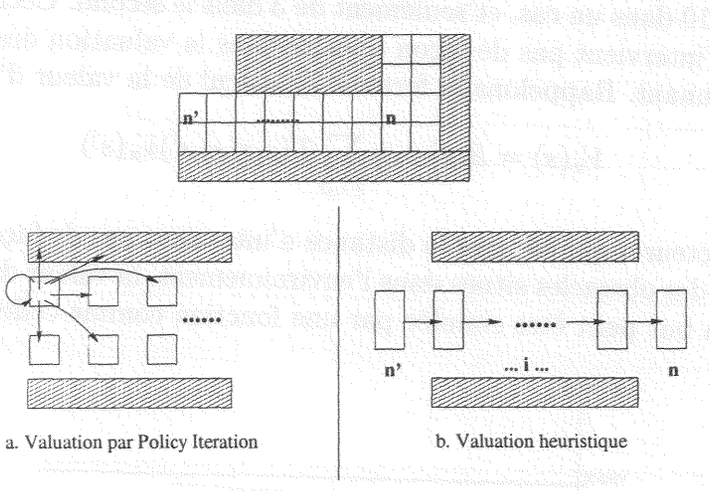


FIG. 5.8 – *Valuation du graphe selon la première méthode.*

La première méthode de valuation que nous présentons ici part du principe que les valeurs données aux états par la politique optimale dépendent de trois critères : la distance au but, la largeur des couloirs à emprunter et les obstacles à éviter. Dans les sous-paragraphes suivants, nous montrons comment nous combinons ces trois facteurs pour obtenir une fonction de valuation du graphe qui permettra d'affecter une valeur à chaque nœud, valeur qui permettra de définir les coûts des buts des MDP partiels. Le principe général de cette méthode est illustré par la figure 5.8 : la valuation d'un nœud n' est calculée récursivement en fonction du nœud suivant n. La valeur d'un état i situé entre les deux nœuds est égale à la valeur de l'état suivant, à laquelle on ajoute le coût de transition entre ces deux états, coût qui sera fonction de la largeur du couloir et des obstacles l'obstruant.

Distance à parcourir

La distance par rapport au but est évidemment un facteur très important dans le calcul de la valeur d'un état. Afin de mesurer son influence, prenons pour exemple la politique optimale présentée figure 5.9.

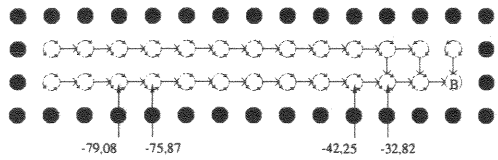


FIG. 5.9 – *Valeur des états en fonction de la distance.*

Le but (état marqué par un 'B', à l'extrême droite de l'environnement) a une valeur nulle (puisque c'est un état absorbant dont le coût est nul). Sur cette figure sont précisées les valeurs de quatre états. Les deux plus proches du but ont une valeur respective de -32,82 et -42,25, alors que les deux plus lointains ont une valeur de -75,87 et -79,08. Dans les deux cas, les états sont situés l'un à côté de l'autre. Pourtant, la différence de valeur est quasiment de 10 dans un cas, et seulement de 3 dans le second. Ceci montre clairement que la distance n'intervient pas de façon linéaire dans la valuation des états, ce qui n'est d'ailleurs pas étonnant. Rappelons la formule de calcul de la valeur d'un état :

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

C'est en fait le facteur γ qui fait que la distance n'intervient pas de façon linéaire. Si on ne tient pas compte des obstacles situés dans l'environnement, la valeur d'un état en fonction de sa distance au but peut être simulée par une fonction comme celle de la figure 5.10.

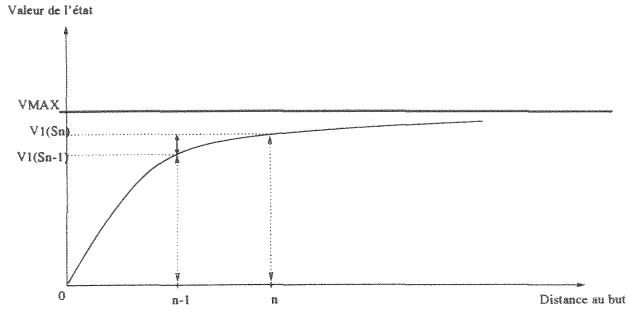


FIG. 5.10 – *Allure générale de la fonction de valeur.*

La valeur d'un état dépend beaucoup de la valeur de ses états adjacents, et plus on s'éloigne du but, plus la valeur augmente (en valeur absolue, rappelons le). Pour simuler ceci, nous utilisons la fonction suivante :

$$V_1(s_n) = V_1(s_{n-1}) + \gamma \frac{\texttt{VMAX} - V_1(s_{n-1})}{\alpha}$$

Dans cette formule, `VMAX` représente la valeur maximale d'un état. Cette valeur est en fait égale à la valeur d'un obstacle $1/(1-\gamma)$. Intuitivement, la formule peut être explicitée ainsi : la valeur d'un état s_n est égale à la valeur de l'état suivant sur le chemin jusqu'au but (s_{n-1}), à laquelle on ajoute une fraction du reliquat de valeur. Ce reliquat représente la valeur qu'il est possible d'ajouter : il est exprimé comme la différence entre la valeur maximale d'un état et la valeur de l'état suivant. Intuitivement, si l'état suivant vaut

70 et que l'intervalle de valuation est [0,100], alors l'état à valuer vaut $70 + x$, où x est une fraction de ce qu'il est possible d'ajouter, c'est-à-dire 30. Ce mode de calcul prend naturellement en compte la non-linéarité de la distance : si s_{n-1} est le but, la transition qui rapproche s_n du but est très importante (dans ce cas $V_1(s_n) = \gamma \times \texttt{VMAX}/\alpha$). En revanche, si s_n est loin du but, la valeur de s_{n-1} est proche de VMAX, donc la valeur de s_n varie peu par rapport à $V_1(s_{n-1})$, ce qui est ce que nous voulons obtenir.

Pour valuer les nœuds de notre graphe, il faut généraliser la formule précédente, puisque les nœuds sont distants de plus d'une transition. La valeur d'un nœud n' est alors égale à la valeur du nœud suivant n, à laquelle s'ajoutent les valeurs des coûts de passage d'un état au suivant sur le chemin reliant les deux nœuds :

$$V_1(n') = V_1(n) + \sum_{i=n}^{i<n'} \gamma \, \frac{\texttt{VMAX} - V_1(i)}{\alpha}$$

Précisons le rôle du facteur α. Le reliquat de valeur à ajouter dépend de la valeur de l'état suivant, mais aussi des obstacles adjacents à ces états. En effet, la fonction de transition étant probabiliste, à chaque passage d'un état à un autre, une collision est possible entre le robot et les murs délimitant un couloir. Le facteur α permet de déterminer le coût de ces collisions. Il dépend à la fois de la largeur du couloir et des obstacles s'y trouvant, et c'est ce que nous allons voir dans le prochain paragraphe.

Obstacles et largeur de couloir

Plus un couloir est large, moins il y a de risque de collision avec les murs le délimitant. Mais si ce couloir contient des obstacles, le risque sera accentué. Le coût de la traversée d'un couloir reliant un nœud n' à un nœud n dépend donc à la fois de sa largeur et du nombre d'obstacles à éviter pour le traverser sans encombres. La valuation du graphe doit donc prendre en compte ces deux facteurs, que nous appelons $l_{n' \to n}$ pour la largeur et $o_{n' \to n}$ pour le nombre d'obstacles. Dans un premier temps, nous avons essayé de les combiner en transformant le facteur α en une fraction de la largeur moyenne du couloir :

$$V_1(n') = V_1(n) + \sum_{i=n}^{i<n'} \gamma \times \frac{\texttt{VMAX} - V_1(i)}{L_{n' \to n}}$$

avec :

$$L_{n' \to n} = \frac{(l_{n' \to n} - 1) \times o_{n' \to n} + l_{n' \to n} \times (d_{n' \to n} - o_{n' \to n})}{d_{n' \to n}}$$

Ici $d_{n' \to n}$ représente la distance entre n' et n. Ainsi, plus le couloir est large, moins la valeur augmente d'un état à l'autre, ce qui est tout à fait réaliste. Mais nous nous sommes rapidement rendu compte que l'utilisation d'une largeur moyenne était un peu naïve. En fait, plus que le nombre d'obstacles, c'est surtout leur configuration qui est importante. Intuitivement, des obstacles situés le long d'un mur sont plus faciles à éviter que le même nombre d'obstacles placés de sorte à former une chicane. C'est ce que nous avons vérifié expérimentalement, grâce aux environnements de la figure 5.11. Cette figure présente deux environnements dans lesquels on a placé six obstacles. Sur la figure de gauche, ces obstacles

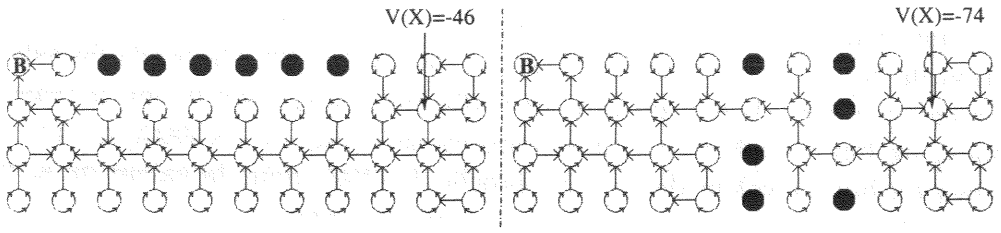


FIG. 5.11 – *Influence des obstacles*

sont placés l'un à côté de l'autre, alors que sur la seconde ils forment une chicane. Dans les deux cas, le but est placé en haut à gauche (état marqué 'B').

La valeur de l'état X est bien plus importante sur la figure de droite (-74 au lieu de -46), alors que la largeur moyenne du couloir est identique. Cette mesure ne pouvait donc nous satisfaire. De plus, cette mesure ne prenait pas en compte la fonction de transition, alors que c'est bien elle qui définit les risques de collision. Pour chaque état du couloir, nous pouvons calculer toutes les transitions amenant à une collision avec un obstacle ; cette somme est ensuite divisée par le nombre d'états contenus dans le couloir, afin de normaliser la valeur. Plus formellement, cette probabilité est exprimée ainsi :

$$r_{n' \to n} = \frac{\sum_{i \in (n' \to n)} \sum_{s_o \in \texttt{suiv\_obs}(i)} T(i, \texttt{Avancer}, s_o)}{N}$$

où N est le nombre d'états contenus dans le couloir reliant n' à n et s_o est un état qui contient un obstacle (ou un mur délimitant le couloir), qui peut être atteint en exécutant une action **Avancer** depuis l'état i. Dans notre nouvelle formule de valuation des arcs du graphe, ce risque de collision se substitue à la largeur moyenne. Notre formule devient alors la suivante :

$$V_1(n') = V_1(n) + \sum_{i=n}^{i<n'} \gamma \, r_{n' \to n} \times \, (\texttt{VMAX} - V_1(i))$$

Ici le risque de collision est un risque moyen, calculé de manière globale sur tout le couloir, et non de façon précise pour chaque état de ce couloir. Ceci a tendance à surévaluer les valeurs des nœuds du graphe, comme on peut le voir sur la figure 5.12, qui représente le graphe obtenu sur notre petit exemple de la figure 5.4. Sur cette figure, les nœuds sont

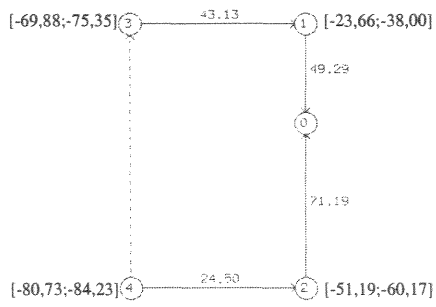


FIG. 5.12 – *Graphe obtenu.*

ordonnés en fonction de leur coût, qui est obtenu en cumulant le coût des arcs permettant d'atteindre le but. Par exemple, le coût du nœud numéroté 4 sur le graphe est de 95,69 (=71,19+24,5). A côté de chaque nœud est indiqué entre crochets la valeur optimale des états contenus dans ce nœud (c'est la valeur affectée à ces états par la politique optimale). On donne un intervalle de valeur car chaque nœud (chaque intersection) contient plusieurs états. On peut remarquer que pour chaque nœud la valeur donnée par notre formule de valuation est bien supérieure à celle donnée par la politique optimale. Comme nous le verrons dans le paragraphe consacré aux résultats, cette surévaluation ne pénalise pas la qualité des solutions obtenues.

Conclusion sur cette méthode

Notre idée de départ, qui était de valuer le graphe en prenant en compte trois critères (distance, obstacles, largeur des couloirs) a sensiblement évolué puisqu'il s'est avéré que les critères de largeur et de présence d'obstacles sont efficacement pris en compte par le risque de collision. Cette formule a l'intérêt d'utiliser la fonction de transition, ce qui la rend robuste aux variations de cette fonction (et aux variations du facteur γ, d'ailleurs). Mais on peut lui reprocher de ne pas prendre en compte la fonction de gains ! Ceci nous a amené à chercher une fonction de valuation qui corrigerait ce défaut ; celle-ci est le sujet du prochain paragraphe.

5.4.2 Seconde méthode de valuation

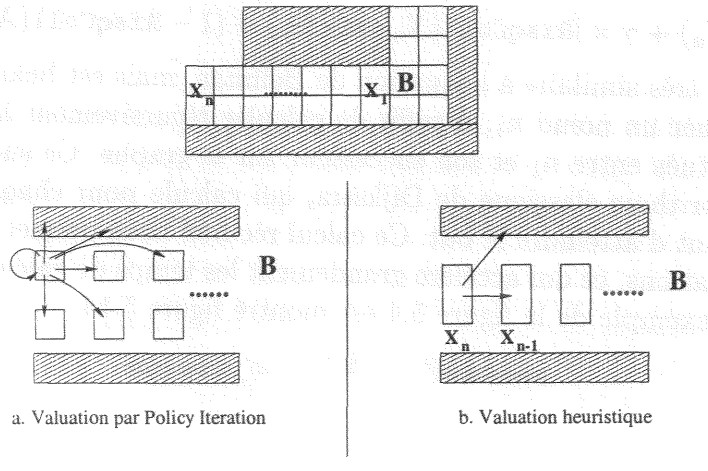


FIG. 5.13 – *Valuation du graphe selon la seconde méthode.*

La fonction de valuation présentée ici est issue des expériences menées pour définir la première formule, présentée ci-dessus. Le but est d'obtenir une fonction de valuation plus proche de la formule de Bellman, plus « naturelle » en quelque sorte. Dans la suite de ce paragraphe, nous nous basons sur l'exemple de la figure 5.13 pour expliquer cette deuxième formule. Le but global de l'environnement est matérialisé par la lettre 'B'. Rappelons tout d'abord comment fonctionne l'algorithme *Policy Iteration* (figure 5.13a). La valeur de chaque état est calculée en utilisant l'équation 2.4 (voir le paragraphe 2.5.3, page 24) :

elle est fonction de la valeur des états qu'il est possible d'atteindre en exécutant une unique action. Ceci est illustré sur la partie gauche de la figure. Chaque état a plusieurs successeurs possibles, il est donc nécessaire de résoudre un système d'équations, ce qui est coûteux. Pour éviter ce problème, nous utilisons une valuation heuristique (figure 5.13b). Parmi les transitions possibles en partant d'un état, on distingue deux possibilités :

1. L'action exécutée a engendré une collision avec un mur ou un obstacle. Les états qui contiennent un obstacle ayant une valeur très négative (voir le paragraphe 3.3.5, page 35), les collisions possibles ont une influence néfaste sur la valeur des états adjacents.
2. L'action a amené le robot dans un état libre de tout obstacle. Dans ce cas, on fait l'hypothèse que les états qu'on peut atteindre ont tous des valeurs similaires.

Ceci nous permet de décomposer la valeur d'un état en deux parties : soit il y a eu une collision, soit il n'y en a pas eu. La probabilité de collision est calculée ainsi :

$$\texttt{RisqColl}(X_n) = \frac{\sum_{s \in X_n} \sum_{s_o \in \texttt{suiv\_obs}(s)} T(s,\texttt{Avancer},s_o)}{|X_n|}$$

Partant d'un état agrégé X_n regroupant tous les états de base nécessaire à représenter la largeur du couloir, on cumule les probabilités de transition vers un état s_o appartenant à l'ensemble $\texttt{suiv\_obs}(s)$ qui contient les états obstrués qu'il est possible d'atteindre en exécutant une action `Avancer` depuis l'état s. Ce cumul de probabilités est divisé par le nombre d'états contenus dans X_n, afin d'obtenir la probabilité moyenne de collision. La fonction de valuation d'un état X_n est alors la suivante :

$$V_2(X_n) = R(X_n) + \gamma \times [\texttt{RisqColl}(X_n) \times V(s_o) + (1 - \texttt{RisqColl}(X_n)) \times V_2(X_{n-1})]$$

Cette formule est très similaire à l'équation de Bellman, mais est beaucoup plus simple à utiliser. Pour valuer un nœud n_1, il suffit de calculer récursivement les valeurs des états X_n,X_{n-1},...,X_0 situés entre n_1 et son successeur sur le graphe. Ce successeur est obtenu en utilisant l'algorithme classique de Dijkstra, qui calcule pour chaque nœud le chemin optimal permettant d'atteindre le but. Ce calcul récursif nous permet d'éviter de calculer un système d'équations, ce qui accélère grandement les temps de calcul. Le graphe obtenu pour notre petit exemple de la figure 5.4 est montré figure 5.14.

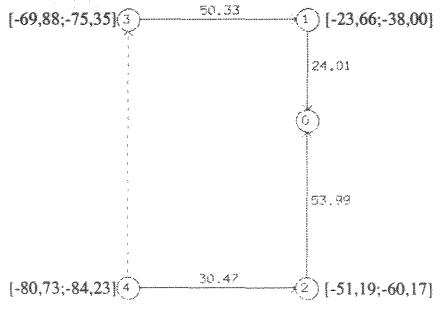


FIG. 5.14 – *Graphe obtenu.*

On peut voir sur cette figure que les valeurs données aux nœuds du graphe sont très proches de celles spécifiées par la politique optimale.

Remarque :

Le risque de collision calculé ici n'est pas le même que celui calculé pour la première méthode de valuation. Ici, ce risque est calculé précisément pour chaque état. Lors du passage d'un état X_n à un état X_{n-1}, on utilise le risque de collision calculé précisément pour X_n, et non un risque moyen commun à tout le couloir. Ceci permet d'estimer le risque de collision de façon plus précise, comme on le voit sur le graphe.

Conclusion sur cette méthode

Cette méthode de valuation du graphe est très naturelle puisqu'elle s'inspire largement de la fonction de valeur de Bellman. Parmi les transitions possibles, on distingue celles amenant à une collision de celles amenant à un état libre. Par rapport à la méthode précédente, celle-ci présente l'avantage d'utiliser la fonction de gain, ce qui la rend plus robuste aux modifications de cette fonction. Nous considérons que cette méthode est à la fois plus naturelle et plus robuste, ce qui ne veut pas dire que la première doit être abandonnée, comme nous le verrons dans le paragraphe de présentation des résultats. Dans le paragraphe suivant, nous expliquons comment ce graphe valué est utilisé pour générer les MDP partiels dont les politiques permettront de former une politique globale.

5.5 Algorithme de construction du graphe

Dans ce paragraphe, nous présentons plus en détail l'algorithme utilisé pour calculer les valeurs des nœuds du graphe. Nous présentons en fait ici deux algorithmes : le premier algorithme (algorithme 5) est une procédure de valuation classique de graphe, calculant la valeur de chaque nœud en fonction de son voisin le plus intéressant (algorithme de Dijkstra). Ici le voisin le plus intéressant n'est pas connu, mais est calculé dynamiquement. On commence par donner une valeur nulle au nœud contenant le but global du MDP, puis on traite ses voisins : leur valeur est calculée à l'aide de l'algorithme 6, décrit ci-dessous. Ensuite, on prend parmi ces voisins celui qui a la valeur la plus faible ; celui-ci devient le nœud courant, qui va nous servir à valuer ces voisins, et ainsi de suite jusqu'à avoir traité tout le graphe.

Le second algorithme (algorithme 6) calcule récursivement la valeur d'un nœud en fonction d'un autre nœud considéré comme son successeur sur le chemin menant au but. Cette valeur est calculée en fonction de la valeur du nœud suivant à laquelle on ajoute le risque de collision sur les états reliant les deux nœuds. Il est à noter qu'ici nous n'avons présenté que la seconde méthode de valuation : l'algorithme de calcul selon la première méthode n'est pas assez différent pour qu'il soit intéressant de le détailler.

Ces deux algorithmes sont donc utilisés non seulement pour calculer les valeurs des nœuds, mais aussi pour déterminer quel type d'arc relie deux nœuds. Ceci est très important dans la suite. En effet, deux nœuds n_1 et n_2 peuvent être reliés soit par un arc dessiné en trait plein, soit par un arc dessiné en pointillés. Dans le premier cas, cela signifie que le meilleur chemin partant de n_1 et menant au but passe par n_2. Dans le second cas, cela signifie que le meilleur chemin partant de n_1 passe par un autre nœud n_3. Dans ce

cas, l'arc entre n_1 et n_2 est représenté en pointillé. Nous verrons au paragraphe suivant comment sont traités ces deux types d'arc.

Ces algorithmes sont tous deux très simples et rapides. La complexité de l'algorithme 5 est liée au nombre de nœuds du graphe, et non plus au nombre d'états du MDP, ce qui est un progrès très important. Seule la procédure de calcul de la valeur d'un nœud par rapport à un autre demande le parcours des états du couloir reliant les nœuds, mais chaque couloir contenant peu d'états, ce calcul reste très rapide, comme nous le vérifierons dans le paragraphe consacré à la présentation des résultats.

5.6 Passage du graphe aux MDP

5.6.1 Construction des MDP partiels

Une fois le graphe obtenu, il nous reste à générer un MDP partiel par région. Sur le graphe, on distingue deux types d'arcs : les arcs en trait plein, et les arcs en traits pointillés. Un arc en trait pointillé est en fait un arc qui ne constitue pas une partie de chemin optimal. C'est le cas de l'arc entre les nœuds 4 et 3 de notre graphe. Depuis le nœud 3, le chemin optimal mène au nœud 1 ; partant du nœud 4, le chemin optimal mène au nœud 2. En revanche, selon la position exacte à l'intérieur de la région représentée par l'arc $4 \rightarrow 3$, il sera optimal de se diriger vers le nœud 3 et pénalisant de choisir le nœud 4. On peut voir sur la figure 5.5 (page 109) que cela correspond tout à fait à ce qui se passe dans le couloir vertical de gauche : tous les états ne mènent pas à la même intersection. Un arc de ce type est donc transformé en MDP à buts multiples, un à chaque extrémité de la région, comme le montre la figure 5.15.

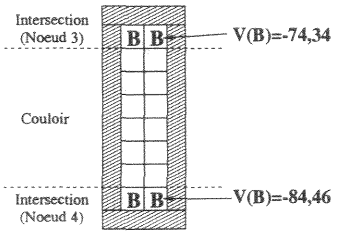


FIG. 5.15 – *MDP partiel correspondant à la région R_5.*

Parmi ces buts, certains doivent être plus attractifs que d'autres pour correspondre au mieux à la politique optimale du MDP global. Comme on peut le voir sur la figure, nous utilisons les valeurs des nœuds du graphe comme valeur des buts à atteindre. Ainsi, les buts situés dans la partie haute du MDP sont légèrement plus attractifs que ceux du bas, ce qui permet d'obtenir un comportement très proche voire identique à celui spécifié par la politique optimale globale.

Le deuxième type d'arc (en trait plein) est traité plus simplement. En effet, tous les états situés sur cet arc (c'est-à-dire dans cette région) mènent au même nœud (c'est-à-dire à la même intersection). C'est le cas par exemple de l'arc entre les nœuds 3 et 1 du graphe. Ceci correspond tout à fait à la politique optimale, comme on peut le vérifier sur la figure 5.5. Dans ce cas, la région correspondante (R_4) est modélisée par un MDP qui a

Algorithme 5 Construction d'un graphe valué représentant un MDP $\mathcal{M}$.

```
VMAX : CONSTANTE
l_noeud : LISTE de NOEUD
noeud_courant,voisin : NOEUD
nb_noeud, orient : ENTIER
nouvelle_valeur : REEL

//aucun noeud n'a encore été traité, leurs valeurs sont nulles
pour tout noeud_courant appartenant à l_noeud faire
  noeud_courant.marque ⇐ FAUX
  noeud_courant.valeur ⇐ VMAX
fin pour

//On commence par traiter le noeud qui contient le but
//Sa valeur est nulle et ne changera pas : on marque le noeud
noeud_courant ⇐ l_noeud.ChercheBut()
noeud_courant.marque ⇐ VRAI
noeud_courant.valeur ⇐ 0
nb_noeud ⇐ 1
//Traitement de tous les noeuds du graphe
tant que nb_noeud <= l_noeud.NombreElements faire
  //Calcul de la valeur des noeuds voisins du noeud courant
  pour orient= 1 à 4 faire
    voisin⇐ChercheVoisin(noeud_courant,orient) //récupération du voisin n° orient
    //si ce voisin existe et que sa valeur n'a pas encore été calculée, on le traite
    si voisin!=0 ET NON voisin.marque alors
      //Appel de la fonction CalculValeur qui calcule la valeur d'un noeud en fonction
      //d'un noeud suivant
      nouvelle_valeur=CalculValeur(voisin,noeud_courant)
      si nouvelle_valeur < voisin.valeur alors
        voisin.valeur ⇐ nouvelle_valeur
      fin si
    fin si
  fin pour

  //Recherche dans la liste du noeud non encore traité de plus petite valeur
  noeud_courant=l_noeud.ChercheMinimum_NonMarque()
  noeud_courant.marque=VRAI
  nb_noeud ++
fin tant que
```

Algorithme 6 Calcul de la valeur d'un nœud en fonction d'un autre

Fonction CalculValeur(n_courant : NOEUD, n_suiv : NOEUD)

VOBS : CONSTANTE //*Valeur d'un état obstrué* $(1/(1-\gamma))$
s : ENTIER
rc : REEL
val_courant,val_precedent : REEL

//*Sauvegarde de la valeur du nœud*
val_precedent=n_suiv.valeur

//*Calcul récursif de la valeur de n_courant*
pour tout s $\in$ (n_courant $\rightarrow$ n_suiv) **faire**
  //*Calcul du risque de collision depuis l'état s du MDP* $\mathcal{M}$
  rc $\Leftarrow$ $\mathcal{M}$.RisqueColl(s)
  val_courant = $\mathcal{M}$.Reward(s) + $\gamma\times$ (rc $\times$ VOBS + (1 - rc) $\times$ val_precedent)
  val_precedent=val_courant
fin pour
renvoyer val_courant

pour buts les états contenus dans l'intersection correspondant au nœud 1. La figure 5.16 montre le MDP partiel construit pour calculer la politique optimale de cette région. Dans la mesure où les buts sont situés à une seule extrémité de la région, leur valeur est fixée classiquement (gain nul d'où une valeur nulle).

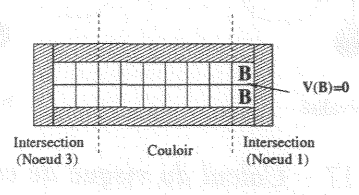


FIG. 5.16 – *MDP partiel correspondant à la région* R_4.

Il ne reste plus ensuite qu'à résoudre séparément chaque petit MDP, puis à combiner les politiques locales pour obtenir une politique globale.

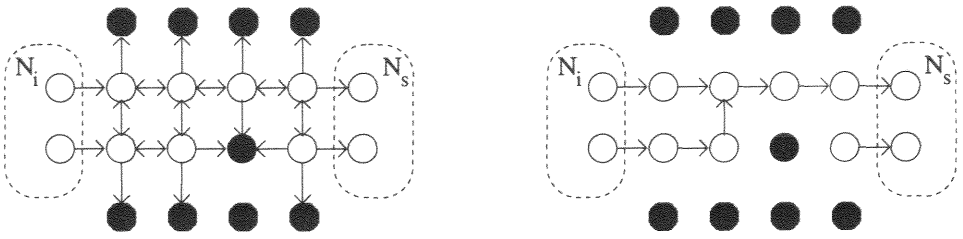
5.6.2 Combinaison des MDP partiels

Chaque MDP partiel est résolu en utilisant l'algorithme *Policy Iteration*. Ensuite, il s'agit de combiner les solutions locales pour obtenir une solution globale. Cela ne pose pas de problème pour les états situés à l'intérieur d'un couloir (ils n'apparaissent que dans un seul MDP partiel). En revanche, les états situés aux intersections peuvent être présents dans deux MDP. Par exemple, certains états du nœud 3 sont englobés à la fois dans le MDP de la figure 5.15 et dans celui de la figure 5.16. Il faut donc faire un choix pour ces états, entre les deux actions optimales calculées dans ces deux MDP. En fait, le choix est immédiat, puisque dans un de ces deux MDP, ces états constituent un but. Un but étant un état absorbant, l'action spécifiée n'est pas utilisable pour la politique globale. Lors du transfert des actions optimales des MDP partiels vers le MDP global, on ignore donc les états qui sont des buts.

5.6.3 Utilisation des MDP partiels pour affiner la formule de valuation

Avant de présenter les résultats obtenus, nous montrons comment les solutions des MDP partiels sont utilisées pour affiner notre façon de calculer le risque de collision. Quelle que soit la méthode de valuation utilisée, ce risque est en fait surévalué par rapport au risque de collision réel, comme on peut le voir sur la figure 5.17.

La partie gauche de cette figure montre comment nous calculons le risque de collision entre deux nœuds N_i et N_s : afin d'assurer la prise en compte de toutes les collisions éventuelles, le risque est calculé en partant du principe que tout état peut amener à une collision, quelle que soit son orientation. La partie droite de la figure montre comment le risque réel pourrait être calculé : si on connaît la politique optimale entre deux nœuds du graphe, on peut calculer le risque de collision réel, car seuls les états dont l'action optimale est d'avancer peuvent amener à une collision. Or nous ne connaissons pas la politique optimale (puisque c'est ce que nous cherchons), mais nous connaissons la politique calculée pour chaque MDP partiel. L'idée ici est d'utiliser ces solutions dans une deuxième phase



a. Risque de collision sur-évalué b. Risque de collision réel

FIG. 5.17 – *Calcul du risque de collision.*

de valuation. Cette deuxième phase a été appliquée uniquement à la seconde méthode de valuation, qui s'y prêtait mieux.

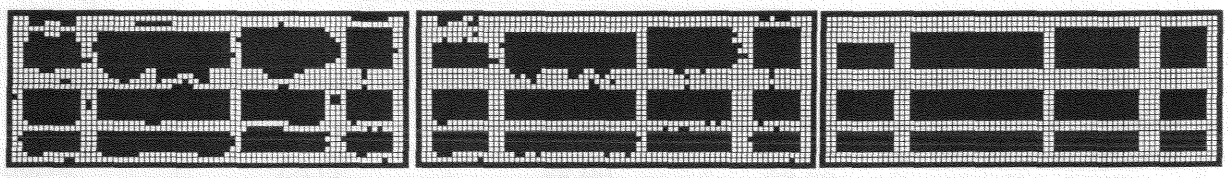
Lors de la première phase de valuation, tous les arcs sont valués et les nœuds ordonnés selon leur valeur, en utilisant sans modification l'algorithme 5. Ensuite, les MDP à but unique, qui correspondent à un arc en trait plein sur le graphe, sont résolus. C'est alors que nous passons à la seconde phase de valuation. Pour ces MDP, nous avons calculé une politique, et nous allons nous servir de cette politique pour ré-évaluer le risque de collision. Ici il est calculé en fonction uniquement des états dont l'action optimale (selon la politique partielle) est d'avancer. L'hypothèse implicite est que la politique partielle est proche de la politique optimale, et donc le risque de collision induit par la politique partielle est similaire à celui induit par la politique optimale. Nous nous servons de ce nouveau risque pour revaluer chaque nœud. Si cette revaluation n'a aucune influence sur les politiques partielles des MDP à but unique, elle permet en revanche d'affiner les valeurs des buts des MDP à buts multiples. Ceux-ci sont donc résolus en fonction de ces nouvelles valeurs, ce qui améliore la qualité des politiques, comme nous le verrons dans le paragraphe suivant qui présente les résultats de nos deux méthodes de valuation.

5.7 Résultats

Dans ce paragraphe, nous présentons les résultats obtenus sur différents environnements, aussi bien en terme de qualité des politiques obtenues qu'en terme de temps de calcul. Nous montrons également la robustesse de notre approche vis-à-vis de différents critères.

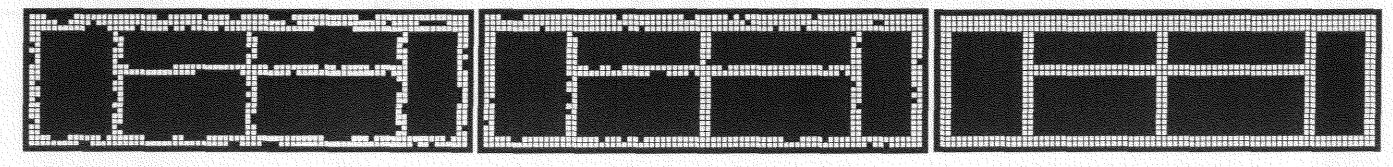
5.7.1 Qualité des solutions

Dans un premier temps, nous présentons des résultats obtenus sur les deux environnements déjà utilisés précédemment, repris figures 5.18 et 5.19. Ici aussi, nous avons voulu tester la robustesse de notre approche en fonction du nombre d'obstacles à éviter. Nous utilisons donc les mêmes 10 instances d'environnement. Toujours en accord avec le chapitre précédent, le but a été placé successivement en 20 positions différentes pour le premier environnement, et 13 sur le second. Dans un premier temps, nous présentons dans les tables 5.1 et 5.2 les résultats globaux, qui sont donc une moyenne sur respectivement



a. 191 obstacles (3360 états) b. 81 obstacles (3800 états) c. Aucun obstacle (4124 états)

FIG. 5.18 – *Premier environnement de test.*



a. 130 obstacles (2192 états) b. 58 obstacles (2480 états) c. Aucun obstacle (2712 états)

FIG. 5.19 – *Deuxième environnement de test.*

200 et 130 problèmes.

Les résultats sont évalués en fonction des deux critères précédemment introduits : la qualité par rapport à la solution optimale et le pourcentage d'états pour lesquels l'action optimale de la solution agrégée est égale à l'action optimale de la solution de base (voir le paragraphe 3.4.1, page 40). Comme précédemment, trois chiffres sont donnés dans les tables 5.1 et 5.2 : la qualité moyenne, à laquelle s'ajoutent les qualités des plus mauvaises et meilleures politiques obtenues parmi tous les problèmes résolus. Nous comparons également nos résultats à ceux donnés par un algorithme *plus court chemin* (présenté au paragraphe 3.4.3, page 48).

Dans chaque table, la première colonne donne les résultats obtenus en utilisant la première méthode de valuation du graphe (paragraphe 5.4.1), alors que la deuxième et la troisième colonne montrent les résultats obtenus par la seconde méthode de valuation. La colonne intitulée « Sans revaluation » correspond à la valuation en une seule phase (paragraphe 5.4.2), alors que celle intitulée « Après revaluation » utilise la valuation en deux phases telle que présentée au paragraphe 5.6.3. Enfin, la dernière colonne présente la qualité des politiques *plus court chemin* correspondantes.

	Valuation 1	Valuation 2		Solution
		Sans revaluation	Après revaluation	*plus court chemin*
Qualité moyenne	95,20	94,95	95,59	65,05
Qualité minimum	84,66	86,45	86,72	26,78
Qualité maximum	98,83	98,75	98,80	88,87
% Action moyen	89,08	87,92	89,56	66,06
% Action minimum	84,74	81,58	81,36	53,73
% Action maximum	92,37	91,52	93,87	73,84

TAB. 5.1 – *Résultats sur l'environnement de la Figure 5.18*

Sur ces deux environnements, on se rend compte que la qualité des politiques obtenues

	Valuation 1	Valuation 2		Solution
		Sans revaluation	Après revaluation	*plus court chemin*
Qualité moyenne	97,97	97,97	98,15	83,27
Qualité minimum	94,32	94,33	94,35	34,00
Qualité maximum	99,89	99,88	99,85	97,60
% Action moyen	89,43	88,41	89,71	74,88
% Action minimum	80,88	78,88	79,97	54,02
% Action maximum	96,12	95,78	95,91	85,54

TAB. 5.2 – *Résultats sur l'environnement de la Figure 5.19*

varie peu, quelle que soit la méthode de valuation utilisée. Nous comparerons plus en détail les méthodes de valuation au paragraphe 5.7.3. Pour le premier environnement, la qualité des politiques va de 94,95% à 95,59% de l'optimal. Ces chiffres sont à comparer avec la qualité moyenne d'une politique *plus court chemin*, qui est seulement de 65%! De plus, contrairement à une politique *plus court chemin* qui peut être très mauvaise dans certains cas (26,78% par rapport à l'optimal dans le pire cas), notre approche ne descend jamais (sur les tests effectués) au-dessous de 84,66%. En terme de pourcentage d'actions identiques, les résultats sont sensiblement les mêmes : le pourcentage moyen est de 89%. Ce chiffre est inférieur à la qualité des politiques, ce qui montre que les erreurs sont faites en majorité pour les états éloignés du but. Les résultats obtenus sur le second environnement de test sont un peu meilleurs. La qualité moyenne atteint plus de 98%, alors que la politique la moins bonne représente plus de 94% de l'optimal (contre seulement 34% pour une politique *plus court chemin*). On peut remarquer néanmoins que la politique *plus court chemin* est également plus proche de l'optimal. En fait cet environnement est plus simple que le précédent (couloirs moins nombreux et généralement moins obstrués), ce qui explique les meilleurs résultats.

Notre méthode de décomposition fait l'hypothèse que les bureaux et halls peuvent être traités comme des cas particuliers de couloirs. Nous avons donc testé notre approche sur l'environnement de la figure 5.20, qui comporte 4 pièces. Les résultats obtenus sur cet environnement, dans lequel le but a été placé en 26 positions successives, sont présentés dans la table 5.3. Ces résultats sont sensiblement identiques aux précédents : notre approche atteint une qualité moyenne de plus de 96%, à comparer aux 75% d'une politique *plus court chemin*. Notre hypothèse de départ est donc tout à fait vérifiée.

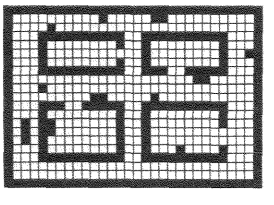


FIG. 5.20 – *Troisième environnement (1640 états).*

	Valuation 1	Valuation 2		Solution
		Sans revaluation	Après revaluation	*plus court chemin*
Qualité moyenne	96,67	96,56	96,78	75,83
Qualité minimum	93,89	93,58	93,75	51,21
Qualité maximum	98,18	98,13	98,17	86,89
% Action moyen	92,68	92,24	93,20	73,84
% Action minimum	90,71	91,14	90,10	67,30
% Action maximum	95,23	95,05	95,11	77,44

TAB. 5.3 – *Résultats pour l'environnement Figure 5.20*

5.7.2 Temps de calcul

Nous venons de voir que la qualité des politiques calculées en utilisant notre approche est globalement très satisfaisante. Reste à voir le gain de temps que cette méthode peut apporter. La table 5.4 contient les temps de calcul moyens obtenus sur nos 3 environnements (356 problèmes au total). Ces valeurs sont des temps CPU exprimés en secondes, et ont été obtenus sur un quadri-processeur Sun Ultra Sparc. Le calcul de la solution optimale a été fait en initialisant *Policy Iteration* avec une politique *plus court chemin*, afin de ne pas injustement pénaliser cet algorithme en partant d'une politique aléatoire. La colonne de gauche présente le temps de calcul total nécessaire pour obtenir une politique approximative en utilisant notre approche (construction du graphe, résolution séquentielle des MDP partiels et combinaison des solutions partielles). Les temps donnés ont été obtenus en utilisant la seconde méthode de valuation (avec revaluation), mais les résultats sont sensiblement identiques pour les autres méthodes de valuation. On voit que plus le MDP global est grand, plus notre approche permet de réduire les temps de calcul. Pour le premier environnement, notre méthode est 52 fois plus rapide que *Policy Iteration* (35 et 26 fois pour le deuxième et troisième environnement). Plus important, ces temps de calcul permettent d'envisager sérieusement une utilisation temps-réel des MDP, puisqu'on passe enfin sous la minute, alors que nos environnements peuvent comporter plus de 4000 états. La décomposition se prêtant naturellement au parallélisme, ces temps de calcul pourraient encore être améliorés : à titre indicatif, le MDP partiel le plus long à calculer pour le premier environnement a nécessité seulement 5 secondes.

	Notre méthode	Solution optimale
Figure 5.18	29	1510
Figure 5.19	18	631
Figure 5.20	5	134

TAB. 5.4 – *Temps de calcul (en secondes).*

5.7.3 Robustesse

Nous avons montré plus haut que nos formules de valuation donnent globalement de bons résultats, quelle que soit la méthode de valuation utilisée. Mais ces tests ont été fait en utilisant une unique fonction de transition, un facteur γ fixé à 0,99 et une fonction de gain très simple. Nous avons voulu tester notre méthode en faisant varier ces facteurs, afin de vérifier sa robustesse.

Obstacles dans l'environnement

Comme nous l'avons indiqué précédemment, les résultats présentés sur les environnements des figures 5.18 et 5.19 sont des résultats moyens sur 10 instances de chaque environnement. La première instance contient un grand nombre d'obstacles, la dernière n'en contient aucun. Nous présentons ici le détail de ces résultats. Tout d'abord, si l'on compare notre approche vis-à-vis d'une politique *plus court chemin* (figures 5.21 et 5.22), on remarque que les politiques obtenues par notre approche sont beaucoup moins sensibles au nombre d'obstacles que les politiques *plus court chemin*. Sur ces deux figures, l'environnement numéro 1 est le plus obstrué, le numéro 10 ne contenant aucun obstacle. Les politiques *plus court chemin* sont moins bonnes dans les environnements sans obstacles, ce que nous avons déjà expliqué. Quand il y a beaucoup d'obstacles dans un couloir, il n'y a pas beaucoup de façons différentes de le traverser : il faut éviter les obstacles, donc la politique *plus court chemin* ne pourra pas s'éloigner beaucoup de la politique optimale. En revanche, s'il n'y a pas d'obstacle, une politique stochastique va privilégier les états au milieu du couloir (minimisant ainsi les risques de collision), ce que ne fait pas une politique *plus court chemin*.

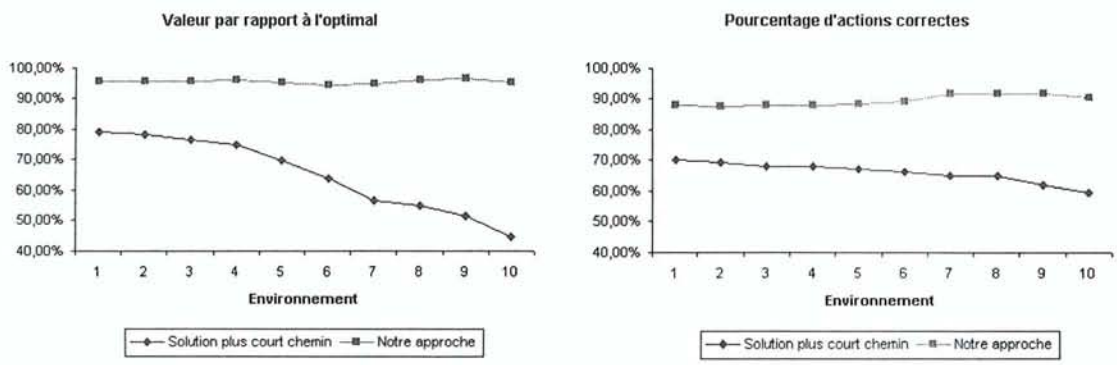


FIG. 5.21 – *Qualité sur les 10 instances de la Figure 5.18*

Si les résultats obtenus avec notre méthode semblent très stables, c'est surtout dû à un effet d'échelle. Les tables 5.5 et 5.6 présentent plus précisément les résultats sur les 10 instances : la colonne la plus à gauche donne les résultats pour l'environnement le plus obstrué, alors que la dixième colonne présente ceux de l'environnement vide de tout obstacle. Les tendances obtenues sur les deux environnements au niveau de la qualité des politiques sont un peu différentes. En effet, sur le second environnement, on perd 1,6% en valeur, mais on gagne quasiment 3% au niveau des actions. Sur le premier environnement,

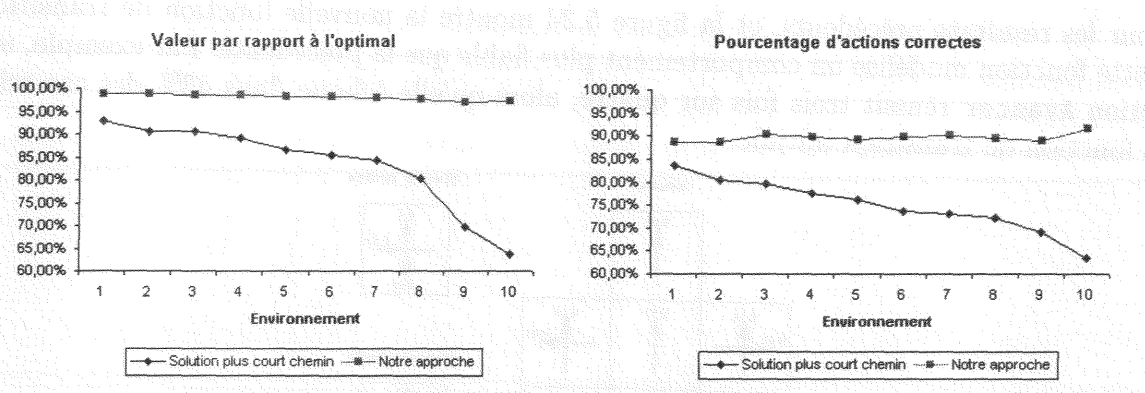


FIG. 5.22 – *Qualité sur les 10 instances de la Figure 5.19*

on gagne 2,5% en actions, alors que la variation en terme de valeur est plus floue. On peut tirer deux conclusions. Premièrement, en terme de pourcentage d'actions identiques, notre méthode est clairement meilleure quand il y a peu d'obstacles. Ceci vient du fait que nos fonctions de valuation surestiment les risques de collision. Plus il y a d'obstacles, plus les valeurs des nœuds se rapprochent de la valeur maximale. En conséquence, les coûts de passage d'une région à une autre sont très similaires pour les régions situées loin du but. En revanche, quand il y a peu d'obstacles, les valeurs des nœuds sont plus faibles et les coûts de passage d'une région à l'autre sont alors définis plus finement, ce qui permet d'obtenir des politiques partielles plus proches de l'optimal. La seconde conclusion à tirer est que l'influence des obstacles est plus floue en terme de valeur des politiques obtenues. Si l'on regarde les résultats sur le second environnement (table 5.6), il semble que la qualité se dégrade au fur et à mesure qu'on retire des obstacles. Mais si l'on regarde les résultats du premier environnement (table 5.5), on ne retrouve pas le même comportement, puisque le meilleur résultat est obtenu sur l'avant-dernière instance. En fait, il semblerait que c'est plus la disposition des obstacles que leur nombre qui influe sur la qualité des résultats.

Qualité	95,85	95,72	95,65	95,99	95,13	94,55	95,07	96,06	96,44	95,45
Actions	87,98	87,59	88,05	88,25	88,65	89,27	91,74	91,90	91,72	90,48

TAB. 5.5 – *Figure 5.18 : détail des résultats sur les 10 instances (après revaluation).*

Qualité	98,89	98,80	98,59	98,46	98,41	98,21	98,14	97,59	97,06	97,35
Actions	88,67	88,78	90,47	89,90	89,28	89,84	90,19	89,47	89,04	91,45

TAB. 5.6 – *Figure 5.19 : Détail des résultats sur les 10 instances (après revaluation).*

Modification de la fonction de transition

Dans ce paragraphe, nous avons effectué les mêmes tests sur nos trois environnements, mais après avoir modifié la fonction de transition. La figure 5.23 montre celle utilisée

pour les résultats précédents, et la figure 5.24 montre la nouvelle fonction de transition. Cette fonction modélise un comportement plus fiable que la précédente. Par exemple, une action **Avancer** réussit trois fois sur quatre, alors qu'elle échoue dans 43% des cas selon la fonction de transition initiale.

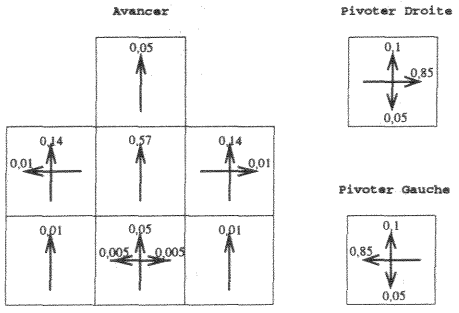


FIG. 5.23 – *Fonction de transition initiale.*

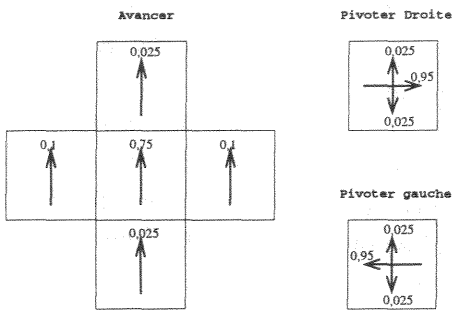


FIG. 5.24 – *Fonction de transition modifiée.*

Les tables 5.7, 5.8 et 5.9 montrent les résultats obtenus sur nos environnements de test. Les résultats sont sensiblement identiques aux précédents, quelle que soit la fonction de valuation utilisée. On remarque en fait que les résultats sont un peu moins bons, mais c'est aussi vrai pour les politiques *plus court chemin*. On peut expliquer cela simplement : notre nouvelle fonction de transition prévoit moins d'échecs (25% au lieu de 43%), donc le choix d'une mauvaise action est plus coûteux.

	Valuation 1	Valuation 2		Solution
		Sans revaluation	Après revaluation	*plus court chemin*
Qualité moyenne	93,43	92,87	94,25	62,51
Qualité minimum	81,65	80,73	83,00	27,45
Qualité maximum	97,56	96,75	97,64	87,85
% Action moyen	85,35	83,46	87,19	64,65
% Action minimum	78,35	75,03	80,77	53,30
% Action maximum	89,45	88,74	91,42	72,79

TAB. 5.7 – *Résultats sur l'environnement de la Figure 5.18 (transitions modifiées)*

	Valuation 1	Valuation 2		Solution
		Sans revaluation	Après revaluation	*plus court chemin*
Qualité moyenne	96,53	96,17	96,38	79,87
Qualité minimum	89,81	89,38	85,64	33,31
Qualité maximum	99,14	98,94	99,04	94,42
% Action moyen	86,11	83,78	87,84	72,53
% Action minimum	76,15	72,26	82,08	53,61
% Action maximum	92,86	90,74	94,80	81,11

TAB. 5.8 – *Résultats sur l'environnement de la Figure 5.19 (transitions modifiées)*

	Valuation 1	Valuation 2		Solution
		Sans revaluation	Après revaluation	*plus court chemin*
Qualité moyenne	95,42	95,11	95,54	74,60
Qualité minimum	92,52	89,97	90,11	45,87
Qualité maximum	97,13	97,13	97,52	87,42
% Action moyen	89,38	88,74	89,96	72,74
% Action minimum	88,14	86,00	86,92	66,26
% Action maximum	91,81	91,01	92,36	76,65

TAB. 5.9 – *Résultats sur l'environnement de la Figure 5.20 (transitions modifiées)*

Modification de γ

Le facteur γ permet d'accorder plus ou moins d'importance aux gains futurs. Il est très important car il permet de régler notamment le comportement vis-à-vis des obstacles. Mais il est plus difficile de le faire varier. S'il est fixé trop faible, les politiques obtenues sont sans intérêt, puisque le but n'est pris en compte que pour les états relativement proches du but. Inversement, un γ très proche de 1 donne de bonnes politiques mais le temps de calcul s'en ressent. En conséquence, nous présentons ici des résultats uniquement sur le troisième environnement (figure 5.20), qui comporte peu d'états (ce qui permet de tester une valeur de γ relativement faible tout en gardant des politiques réalistes et de tester des valeurs fortes en un temps de calcul raisonnable). Les tables 5.10 et 5.11 contiennent les résultats obtenus pour les valeurs : $\gamma = 0{,}95$ et $\gamma = 0{,}999$.

Comparés aux résultats obtenus avec un facteur γ fixé à 0,99 (table 5.3, page 125), ces résultats sont également assez similaires. Quelle que soit la formule de valuation utilisée, elle semble robuste. Selon la valeur de γ, les résultats sont un peu meilleurs ou un peu moins bons, mais les stratégies *plus court chemin* varient dans le même sens. Un facteur γ à 0,95 réduit les variations possibles de valeurs, donc les erreurs ont moins de conséquences négatives (et inversement pour 0,999). On peut noter tout de même que la seconde phase de valuation n'améliore pas toujours les performances : les résultats obtenus avec $\gamma = 0{,}95$ passent de 98,08 à 97,95.

	Valuation 1	Valuation 2		Solution
		Sans revaluation	Après revaluation	*plus court chemin*
Qualité moyenne	97,80	98,08	97,95	92,34
Qualité minimum	96,64	97,23	96,65	80,81
Qualité maximum	98,80	98,94	98,70	96,99
% Action moyen	90,10	92,44	90,85	81,38
% Action minimum	87,69	90,10	87,78	76,47
% Action maximum	92,79	94,19	93,40	83,68

TAB. 5.10 – *Résultats sur l'environnement de la Figure 5.20 ($\gamma = 0{,}95$)*

	Valuation 1	Valuation 2		Solution
		Sans revaluation	Après revaluation	*plus court chemin*
Qualité moyenne	94,21	94,20	95,05	66,46
Qualité minimum	90,82	90,73	91,09	39,56
Qualité maximum	97,32	97,46	97,76	80,94
% Action moyen	89,42	88,97	91,47	69,80
% Action minimum	87,47	86,25	88,45	63,45
% Action maximum	92,79	92,11	93,40	74,51

TAB. 5.11 – *Résultats sur l'environnement de la Figure 5.20 ($\gamma = 0{,}999$)*

5.7.4 Comparaison des fonctions de valuation

Nous avons présenté un certain nombre de tableaux, chacun donnant les résultats obtenus en utilisant nos deux méthodes. Est-il possible d'en tirer des conclusions quant à la meilleure méthode à utiliser? Si l'on survole ces tableaux, on peut en tirer l'impression générale que la seconde formule après revaluation est meilleure que la première qui elle-même est meilleure que la seconde sans phase de revaluation. Mais certains tableaux montrent qu'il existe des contre-exemples. Suivant les paramètres utilisés, il peut arriver que la première formule de valuation soit meilleure que la seconde, voire que la phase de revaluation désavantage la seconde méthode. Nous ne tirerons donc pas de conclusion sur l'efficacité de telle méthode par rapport à une autre. En revanche, les figures 5.25 et 5.26 qui comparent les deux méthodes de valuation sur les deux premiers environnements de test semblent montrer que la seconde méthode donne de meilleurs résultats que la première lorsque les environnements comportent peu d'obstacles. Une combinaison de ces deux méthodes selon ce critère pourrait peut-être donner des résultats intéressants.

Ceci étant dit, ces deux méthodes peuvent également être comparées selon d'autres critères, et notamment selon la robustesse. Nous avons vu précédemment que les deux méthodes sont robustes à la présence d'obstacles et aux modifications de la fonction de transition et du facteur γ. En revanche, les modifications de la fonction de gain rendent la première méthode beaucoup moins efficace puisque celle-ci ne tient pas compte de cette fonction, contrairement à la seconde. Si les résultats obtenus par les deux méthodes sont sensiblement identiques, il n'en reste pas moins que la seconde est plus naturelle puisque

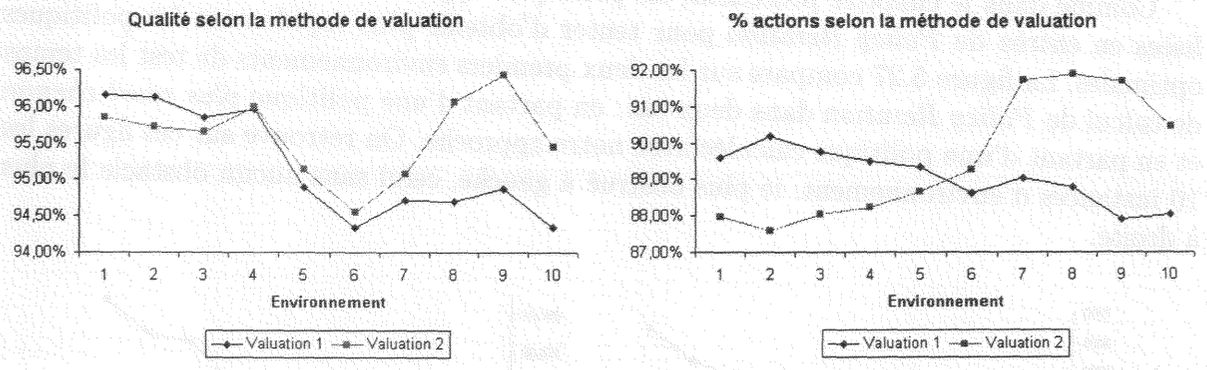


FIG. 5.25 – *Comparaison des 2 valuations sur l'environnement de la figure 5.18*

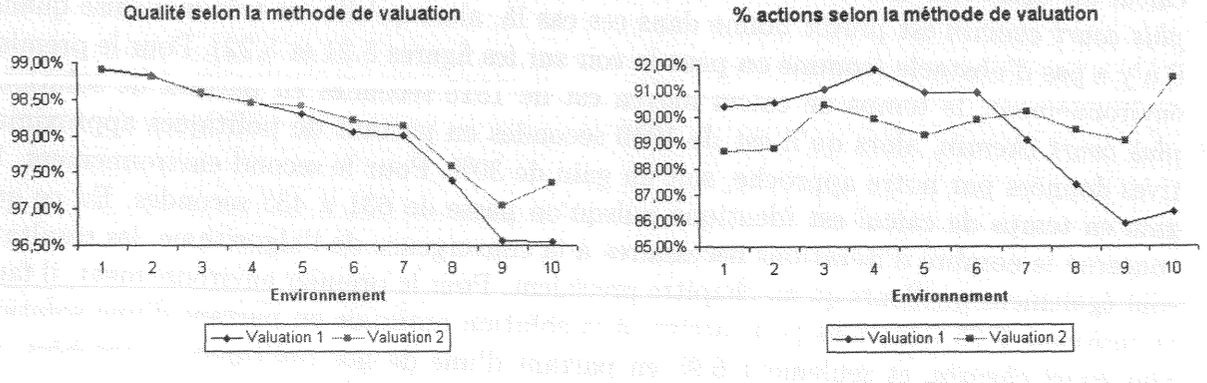
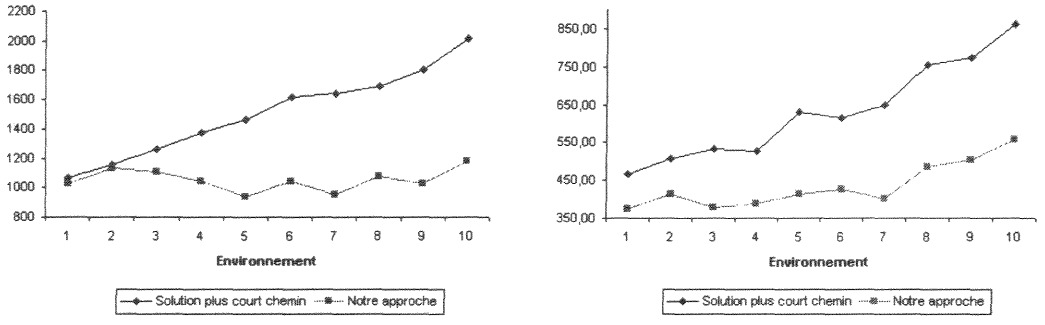


FIG. 5.26 – *Comparaison des 2 valuations sur l'environnement de la figure 5.19*

plus proche de la formule de Bellman.

5.8 Calcul des solutions optimales

Comme dans le chapitre précédent, les politiques approchées calculées ici ont été utilisées en entrée de *Policy Iteration* pour tenter d'obtenir plus rapidement les politiques optimales. La figure 5.27 compare sur les deux premiers environnements de test les temps de calcul de *Policy Iteration* dans deux cas : en partant d'une politique *plus court chemin* et en partant d'une politique calculée avec notre approche. On retrouve sur ces figures les 10 instances d'environnement, le plus obstrué à gauche, celui sans aucun obstacle le plus à droite.



a. Premier environnement

b. Deuxième environnement

FIG. 5.27 – *Temps de calcul pour les figures 5.18 et 5.19.*

Ici les résultats sont bien plus satisfaisants qu'au chapitre précédent, même si le temps-réel n'est pas encore de mise ! Tout d'abord, on peut remarquer que le gain en temps de calcul est quasi nul dans les environnements obstrués. Ceci vient du fait que la politique *plus court chemin* est plutôt bonne dans ces cas là, alors qu'elle est très mauvaise quand il n'y a pas d'obstacle (comme on peut le voir sur les figures 5.21 et 5.22). Pour le premier environnement, le temps de calcul moyen est de 1510 secondes en partant de solutions *plus court chemin*, alors qu'il est de 1050 secondes en partant de politiques approximatives données par notre approche, soit un gain de 30%. Pour le second environnement, le gain en temps de calcul est identique puisqu'on passe de 631 à 433 secondes. En ce qui concerne le nombre d'itérations nécessaires à la convergence de l'algorithme, les résultats sont également meilleurs qu'au chapitre précédent. Pour le premier environnement, il faut en moyenne 9,65 itérations pour arriver à la solution optimale en partant d'une solution *plus court chemin*, et seulement 6,95 en partant d'une de nos politiques approchées, et pour le second environnement on passe de 7,54 à 5,26 itérations.

5.9 Conclusion et perspectives

Dans ce chapitre, nous avons présenté une méthode originale de décomposition. Celle-ci est appliquée à des environnements intérieurs structurés contenant des couloirs, des

bureaux, des halls, etc. Tout ce qu'elle nécessite est une description de l'environnement dans laquelle les états sont étiquetés selon leur appartenance à une intersection ou un couloir (les bureaux et halls étant traités de façon similaire). Le point fort de notre approche tient dans le fait qu'elle définit finement les coûts de passage entre partitions de l'espace d'états, point qui est souvent négligé dans les approches similaires. La représentation de l'environnement à l'aide d'un graphe est simple et intuitive, et peut s'appliquer à de nombreux problèmes. Les politiques obtenues sont très proches de l'optimal, quels que soient les paramètres utilisés, et les temps de calcul très rapides permettent enfin d'envisager une utilisation temps-réel des Processus de Décision Markovien en planification.

Nous sommes globalement très satisfaits de cette approche et la considérons comme le point fort de cette thèse. Néanmoins, ceci pourrait être encore amélioré, et nous envisageons pour ce faire plusieurs suites à ce travail :

- Nous avons vu que l'utilisation des politiques approximatives en entrée de *Policy Iteration* n'est pas très efficace, mais cela ne veut pas dire que les politiques calculées ici ne peuvent pas être utilisées efficacement pour obtenir des politiques optimales. Par exemple, la transformation des politiques définies pour les MDP partiels en macro-actions, comme cela est fait dans [Parr, 1998a] pourrait être plus intéressante.
- Une approche hiérarchique serait également à étudier (figure 5.28). La première phase serait inchangée : représentation de l'environnement à l'aide d'un graphe. Mais, une fois les nœuds ordonnés, seuls certains MDP partiels seraient résolus : sur notre exemple, ceux construits sur les régions constituées par les arcs $\{1 \rightarrow B$, $2 \rightarrow B$, $3 \rightarrow 1\}$. Les solutions de ces MDP partiels seraient ensuite utilisées pour résoudre rapidement le MDP M_1 constitué par la réunion des 3 couloirs. Ensuite, le graphe pourrait être re-valué (pour les nœuds 4 à 8), en fonction des nouvelles valeurs des nœuds 1 à 3. Puis un autre MDP pourrait être résolu (par exemple celui constitué par les nœuds {1,3,4,5}), et ainsi de suite jusqu'à englober tout l'environnement. Cette approche devrait rester rapide, tout en améliorant la qualité des politiques obtenues.

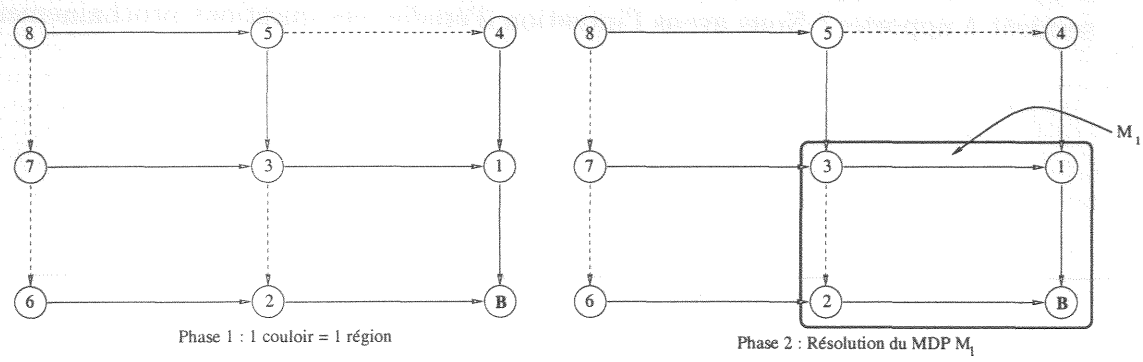


FIG. 5.28 – *Résolution hiérarchique : phases 1 et 2.*

- Lorsqu'on résout un MDP, c'est sur un environnement préalablement fixé, pour atteindre un but également choisi *a priori*. Si on ajoute un obstacle dans l'environnement ou si on déplace le but de 5 mètres, il faut recalculer toute la politique. Or il est clair que de nombreuses parties des politiques seront identiques. La dé-

composition permet naturellement de réutiliser des parties de politique, et notre représentation en graphe y est très adaptée. Prenons l'exemple de la figure 5.29. Un problème source (à gauche) a été résolu, et on cherche à résoudre un MDP dans le même environnement, dans lequel le but a été déplacé. A partir des graphes de résolution des deux MDP, l'utilisation de techniques empruntées au raisonnement à partir de cas (RàPC) pourrait nous aider à résoudre le second MDP. Si certains MDP partiels peuvent être *a priori* réutilisés tels quels (M_3), d'autres doivent être adaptés (M_2 dans le couloir vertical de droite et éventuellement, mais uniquement à condition que les deux couloirs verticaux partagent certaines caractéristiques, dans le couloir vertical de gauche). Le couloir horizontal du bas semble plus difficile à résoudre en utilisant le MDP source, mais la résolution de quelques MDP partiels n'est pas coûteuse, comme nous l'avons déjà vu. Cette piste en est encore à l'état embryonnaire, nous n'avons pas encore d'idée très précise quant aux mécanismes d'adaptation à mettre en œuvre, mais cela nous semble très intéressant.

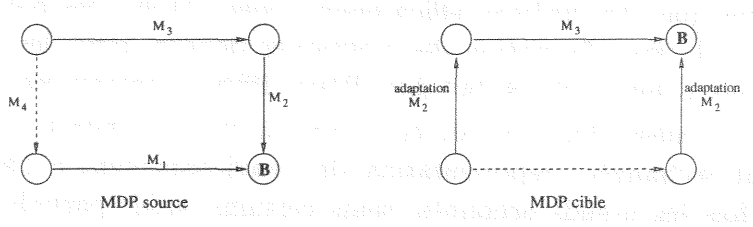


FIG. 5.29 – *Solution d'un MDP cible à partir de la solution d'un MDP source.*

– Nous aimerions tester notre approche sur d'autres types de problèmes que ceux étudiés ici, afin de mieux cerner son niveau de généralité. Notre problème de planification en environnement intérieur structuré se prête naturellement à la décomposition et à la représentation par un graphe. De plus, des états adjacents ont des valeurs similaires, ce que nous utilisons pour valuer notre graphe. Serait-ce aussi simple d'utiliser cette approche pour résoudre d'autres problèmes ? Quelles adaptations seraient à apporter ? Nous avons l'intention d'étudier ces questions prochainement.

Chapitre 6

Bilan des travaux d'agrégation et de décomposition

6.1 Introduction

Dans chaque chapitre, nous avons présenté les résultats obtenus par nos méthodes de calcul de politiques sous-optimales. Nous nous sommes attachés à utiliser à chaque fois les mêmes environnements, afin de pouvoir comparer les mérites respectifs des méthodes développées. Ce chapitre est en fait un bilan de nos trois années de thèse, et reflète bien la progression de notre travail. Partant d'une étude préliminaire des Processus Décisionnels de Markov et de leurs algorithmes de résolution, nous avons fait le constat qu'il était indispensable d'utiliser des techniques heuristiques, afin d'obtenir de « bonnes » politiques sous-optimales en un temps de calcul réaliste. Nous avons d'abord étudié l'apport de petits algorithmes très simples au niveau de l'initialisation de *Policy Iteration* (voir le chapitre 3), puis nous nous sommes aperçus que bien que l'utilisation de ces algorithmes soit beaucoup plus efficace que l'utilisation de politiques aléatoires, le gain en temps de calcul n'était pas suffisant. Nous nous sommes alors tournés successivement vers les techniques d'agrégation d'états et de décomposition, qui nous ont permis d'obtenir des résultats très intéressants, que ce soit en terme de qualité des politiques obtenues ou en terme de temps de calcul. Nous allons rappeler l'intérêt de chaque méthode et comparer leurs résultats respectifs dans la suite de ce court chapitre.

6.2 Qualité des politiques

Les tables 6.1, 6.2 et 6.3 reprennent les résultats des algorithmes que nous avons développés au cours de cette thèse, sur les environnements vus précédemment. Les deux premières colonnes de chaque table présentent les résultats des petits algorithmes présentés au chapitre 3, et les deux dernières montrent les résultats des méthodes d'agrégation et de décomposition. Dans ce cas, nous avons repris uniquement la meilleure méthode développée. Nous ne parlerons pas ici de la quatrième méthode d'agrégation, puisqu'elle est assez différente des autres, et surtout parce que ses résultats ne peuvent être comparés aux autres, puisque les hypothèses de départ sont radicalement différentes.

Passons rapidement sur les deux premières colonnes des tables de résultats. Ce que nous pouvons noter, c'est qu'il est facile d'obtenir des politiques de bonne qualité en moyenne, mais que ces méthodes trop simples peuvent également donner des résultats très mauvais dans certaines configurations. De plus, l'ajout d'une modification mineure de l'algorithme *plus court chemin* permet d'améliorer assez sensiblement les résultats (respectivement 9%, 5,6% et 9,6% pour les trois environnements), mais reste toujours très peu efficace dans certains cas.

Les deux colonnes de droite sont plus intéressantes. Tout d'abord, on note que la décomposition est plus efficace que l'agrégation d'états. En moyenne (améliorations de 5,4%, 2,85% et 9,4%), mais aussi, et c'est tout aussi important, en ce qui concerne les plus mauvais résultats. Si l'algorithme d'agrégation peut encore donner des politiques de qualité plus que moyenne (74,83%, 78,14% et surtout 47,43% sur le troisième environnement), la décomposition permet d'obtenir dans la plupart des cas des politiques de bonne qualité, puisque sur les 356 problèmes traités, on ne descend jamais au-dessous de 86,72%. De plus, le troisième environnement, composé de couloirs et de bureaux, est très mal traité par l'agrégation, alors que la décomposition reste très efficace.

Ces commentaires sont tout de même assez relatifs, car nous n'avons jamais répondu à une question qui serait pourtant intéressante : à partir de quel pourcentage de la politique optimale a-t-on une « bonne » politique. 70% est-il un chiffre suffisant ? 80% ? Difficile de répondre à ces questions. Seule l'exécution des politiques pourrait nous éclairer, mais de nombreux tests seraient nécessaires pour obtenir des résultats significatifs. On peut néanmoins remarquer que ceci est relatif à l'environnement considéré. Nous avons toujours expliqué nos méthodes sur des petits environnements, sans donner les résultats obtenus. Tout simplement parce qu'en-deçà d'une certaine taille d'environnement, les résultats seront « bons » quelle que soit la méthode employée ! On voit bien sur les trois environnements que certains donnent de meilleurs résultats que d'autres, sans que cela soit lié à la taille. Nous pensons tout de même avoir généré des environnements suffisamment difficiles pour pouvoir affirmer que les chiffres que nous donnons sont de bonne qualité. L'idéal aurait été de disposer, comme c'est le cas en Reconnaissance Automatique de la Parole, de corpus de tests permettant de comparer ces résultats à ceux de la communauté. Malheureusement, ce n'est pas le cas dans notre domaine, au contraire, les environnements de travail sont souvent très simplistes [Parr, 1998a; Precup et Sutton, 1997; Hauskrecht *et al.*, 1998].

	plus court chemin	*plus court chemin adapté*	Agrégation	Décomposition
Qualité moyenne	65,05	74,11	90,21	95,59
Qualité minimum	26,78	38,71	74,83	86,72
Qualité maximum	88,87	93,12	98,06	98,80
% Action moyen	66,06	74,20	82,86	89,56
% Action minimum	53,73	65,27	77,90	81,36
% Action maximum	73,84	79,58	90,93	93,87

TAB. 6.1 – *Bilan pour l'environnement de la figure 5.18*

	plus court chemin	*plus court chemin adapté*	Agrégation	Décomposition
Qualité moyenne	83,27	88,87	95,30	98,15
Qualité minimum	34,00	44,63	78,14	94,35
Qualité maximum	97,60	99,47	99,50	99,85
% Action moyen	74,88	79,90	83,46	89,71
% Action minimum	54,02	63,57	77,83	79,97
% Action maximum	85,54	86,45	93,14	95,91

TAB. 6.2 – *Bilan pour l'environnement de la figure 5.19*

	plus court chemin	*plus court chemin adapté*	Agrégation	Décomposition
Qualité moyenne	75,83	85,41	87,32	96,78
Qualité minimum	51,21	72,12	47,43	93,75
Qualité maximum	86,89	94,11	98,03	98,17
% Action moyen	73,84	82,1	86,44	93,20
% Action minimum	67,30	75,37	82,64	90,10
% Action maximum	77,44	86,55	89,98	95,11

TAB. 6.3 – *Bilan pour l'environnement de la figure 5.20*

6.3 Robustesse

Nous avons déjà étudié la robustesse de nos méthodes, surtout en ce qui concerne la méthode de décomposition, dans la mesure où les heuristiques employées nous faisaient craindre d'avoir employé des méthodes trop spécifiques à notre modèle. La méthode d'agrégation étant moins sujette à ces problèmes, nous ne l'avons testée qu'en fonction du niveau d'obstruction des couloirs. On retrouve sur les figures 6.1 et 6.2 les dix instances de nos deux premiers environnements de test (comme précédemment, la plus obstruée à gauche, celle sans obstacle à droite).

Même si la méthode de décomposition est sujette à quelques variations, on peut constater qu'elle est beaucoup plus stable que la méthode d'agrégation, puisque l'efficacité de cette dernière varie selon l'obstruction. Nous avons déjà expliqué cela : un suivi d'un chemin sous-optimal est beaucoup plus grave selon que ce chemin est obstrué ou qu'il est au contraire large et facile à traverser. Mais l'évolution loin d'être uniforme montre aussi que plus que le niveau d'obstruction des couloirs, c'est surtout la configuration des obstacles qui peut poser problème. Notre façon de calculer le risque de collision (quelle que soit la méthode) n'est qu'approximative, et les résultats s'en ressentent quelque peu.

En conclusion, il faut noter que la décomposition est globalement moins sensible aux obstacles que l'agrégation, mais que cette dernière peut être pratiquement aussi efficace lorsque les environnements comportent très peu d'obstacles.

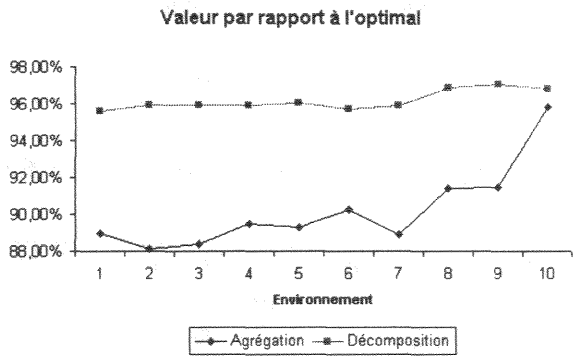


FIG. 6.1 – *Qualité sur les 10 instances de la Figure 5.18*

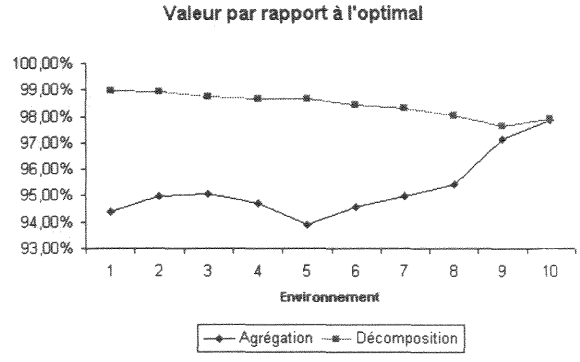


FIG. 6.2 – *Qualité sur les 10 instances de la Figure 5.19*

6.4 Temps de calcul

	Solution optimale	Agrégation	Décomposition
Figure 5.18	1510	156	29
Figure 5.19	631	57	18
Figure 5.20	134	23	5

TAB. 6.4 – *Temps de calcul (en secondes).*

La table 6.4 reprend les temps de calcul moyens obtenus avec nos approches. La colonne de gauche donne le temps de référence, c'est-à-dire le temps de calcul nécessité par *Policy Iteration*, que nous avons initialisé avec une politique *plus court chemin*. Viennent ensuite les temps de calcul des méthodes d'agrégation et de décomposition. Si les deux méthodes permettent de gagner beaucoup de temps par rapport à la solution de référence, il faut noter que la méthode de décomposition est de loin la plus rapide. De plus, elle est beaucoup moins sensible au nombre d'états de l'environnement. L'efficacité de la décomposition dépend surtout du nombre de nœuds du graphe et beaucoup moins du nombre d'états, puisque ceux-ci interviennent uniquement lors du calcul de risque de collision. En revanche, ceci n'est que partiellement vrai pour l'agrégation d'états. La phase de transformation des fonctions de transition dépend du nombre d'états à traiter, et seuls les états des couloirs donnant lieu à des états abstraits, nous avons encore des MDP de bonne taille à résoudre, d'où les temps de calcul moins intéressants.

6.5 Conclusion

Si les politiques calculées par les petits algorithmes *plus court chemin* ou *plus court chemin adapté* peuvent être très efficaces dans des cas très particuliers, il est bien évident qu'on ne peut s'en satisfaire dans le cas général. En revanche, les méthodes d'agrégation et de décomposition sont plus intéressantes, puisqu'elles conservent l'aspect stochastique des politiques. Au vu des résultats, aussi bien en terme de qualité que de robustesse ou de temps de calcul, la décomposition est clairement plus intéressante, dans nos types d'environnement tout au moins. Les hypothèses préalables à chaque méthode y sont évidemment pour beaucoup : l'affectation d'une unique action à un groupe d'états (géographiques) engendre forcément une sous-optimalité. Cela explique sans doute pourquoi dans la littérature consacrée à l'agrégation d'états, les états du MDP sont des données symboliques, représentent plutôt une « situation » du monde que des coordonnées spatiales. Si l'état du processus est $\{$*Robot dans le couloir X, Robot tient un café, Il pleut*$\}$, il est clair qu'on peut agréger ce type d'état en $\{$*Robot dans le couloir X*$\}$ sans perdre d'information, si on s'intéresse uniquement aux déplacements du robot. En revanche, la structure de nos environnements se prête naturellement à la décomposition, et la perte d'information est beaucoup moindre, puisque le calcul heuristique des valeurs des buts de chaque MDP partiel permet de d'approcher les valeurs du MDP global. Enfin, et c'est le résultat qui

nous satisfait le plus, les temps de calcul de cette méthode sont très rapides, et permettent enfin une utilisation temps-réel des politiques stochastiques.

Chapitre 7

Conclusion et perspectives

Pour conclure ce mémoire, nous commencerons par un bref résumé soulignant les apports, avant d'indiquer les différentes perspectives que nous aimerions développer.

7.1 Résumé et apports

Nous avons présenté dans cette thèse une étude de l'utilisation de modèles stochastiques pour planifier les trajectoires d'un robot mobile évoluant dans un environnement intérieur structuré. Lors de mon arrivée en D.E.A., les Processus Décisionnels de Markov étaient encore peu connus dans la communauté *Intelligence Artificielle*, et l'arrivée récente du robot Gaston dans l'équipe RFIA a motivé l'étude de ces modèles, bien adaptés à la prise en compte des diverses imprécisions et incertitudes rencontrées dans une application de robotique mobile. Si dans un premier temps nous nous sommes surtout intéressés au modèle le plus général (les Processus Décisionnels de Markov Partiellement Observés ou POMDP), qui permettent de prendre en compte toutes les incertitudes dès la phase de planification, nous avons rapidement restreint notre champ d'étude à un cas particulier de ces modèles : les Processus Décisionnels de Markov Parfaitement Observés (MDP). En effet, la complexité des algorithmes développés dans le cadre général des POMDP est telle que ceux-ci ne peuvent être appliqués que sur des environnements de petite taille, alors que ceux utilisés dans les applications de robotique mobile nécessitent rapidement la représentation de quelques milliers d'états.

Notre apport dans le cadre des MDP se situe à plusieurs niveaux :

- Dans le chapitre 3, nous montrons l'importance de l'adaptation des différents paramètres afin d'obtenir des politiques intéressantes dans le cadre de la robotique mobile. En effet, si on utilise des MDP, c'est pour obtenir des politiques dont l'exécution est bien sûr rapide, mais également pour obtenir des politiques sûres. Résoudre un MDP pour obtenir une politique dont le seul critère d'optimalité est la distance n'a aucun intérêt. Nous montrons donc dans ce chapitre quels facteurs sont importants pour obtenir ce que nous appelons des politiques intéressantes pour la robotique : à la fois efficaces et sûres, n'hésitant pas à préconiser des détours pour éviter des chemins obstrués qui mettraient en danger le robot, compromettant ainsi le succès des missions allouées. La fonction de transition doit refléter au mieux les

comportements réels du robot, les obstacles doivent être suffisamment répulsifs, et le facteur d'atténuation γ fixé avec soin. Enfin, dans ce même chapitre, nous comparons sur un petit exemple le fonctionnement des deux algorithmes généralement utilisés pour résoudre les MDP, ce qui ne nous semble pas inintéressant puisque la supériorité d'un algorithme sur l'autre n'a pas encore été prouvée.

- Dans le chapitre 4, nous décrivons les méthodes d'agrégation d'états que nous avons développées. En effet, la complexité des MDP est très liée au nombre d'états du processus, ce qui rend l'utilisation tels quels des algorithmes de résolution un peu illusoire dans le cadre d'une application temps-réel. Nous avons proposé quatre méthodes d'agrégation d'états, toutes utilisant les particularités de notre application, à savoir qu'un couloir est un candidat naturel à la création d'un état abstrait, englobant les états de base le constituant. En effet, il n'est pas vraiment nécessaire de représenter finement les états d'un couloir, à condition que l'état abstrait les remplaçant inclue certaines caractéristiques : longueur et largeur du couloir, mais aussi risque de collision à l'intérieur de ce couloir. Si les trois premières méthodes s'attachent avec plus ou moins de succès à obtenir des politiques approximatives proches de l'optimal au sens où on l'entend habituellement, c'est-à-dire en terme de valeur de la politique générée, la quatrième est quant à elle sans doute plus originale. En effet, celle-ci part du principe que puisque l'exécution d'une politique nécessite l'utilisation d'un module de navigation, autant profiter au maximum de ce dernier. Ainsi, certains comportements tels que la navigation au milieu des couloirs (dans une « zone de sécurité », le plus loin possible des murs et obstacles) et l'évitement d'obstacles, qui sont générés par le module de navigation, n'ont pas à être traités par l'algorithme de planification. De ce fait, les couloirs peuvent être agrégés sans tenir compte ni de leur largeur ni du risque de collision. Cette méthode permet d'obtenir des politiques très sous-optimales au sens habituel du terme, mais néanmoins très efficaces, tant que le navigateur est capable de suppléer aux manques de la politique. Nous avons montré que de très bons résultats peuvent être obtenus dans des environnements complexes mais ne contenant qu'un petit nombre d'obstacles, mais qu'il existe des configurations très problématiques pour l'application de cette méthode. Néanmoins les techniques développées permettent de réduire les temps de calcul des politiques, et constituent un premier pas vers l'utilisation des MDP en temps-réel. Ces recherches ont donné lieu à la publication de deux articles dans des conférences internationales [Laroche et Charpillet, 1998; Laroche *et al.*, 1999b].

- Dans le chapitre 5, nous présentons nos travaux dans le cadre de la décomposition de MDP. Nous avons développé deux techniques, toutes deux fondées sur l'utilisation d'un graphe valué pour représenter l'environnement et calculer les valeurs des coûts de passage d'une région de l'environnement à l'autre. Chaque couloir donne lieu à une région de l'environnement, qui communique avec les autres régions via les intersections de l'environnement, qui servent de buts à atteindre. Les nœuds du graphe sont constitués par les intersections de l'environnement, les couloirs ou pièces donnant lieu aux arcs du graphe. Les coûts de passage d'un nœud à l'autre sont calculés à l'aide d'heuristiques, les nœuds étant ordonnés en utilisant un algorithme

classique de *Dijkstra*. La première méthode utilise une heuristique fondée sur le fait que la valeur d'un état est fonction de sa distance au but, mais aussi des obstacles et de la largeur des couloirs qu'il faut emprunter à partir de cet état pour arriver au but. L'heuristique utilisée par la seconde méthode est plus simple et plus naturelle, puisqu'elle est constituée par une adaptation de l'équation de *Bellman*, traitant les transitions différemment selon qu'elles mènent à une collision ou au contraire à un état libre. Les deux méthodes présentées donnent toutes deux d'excellents résultats : politiques très proches de l'optimal, en un temps de calcul très faible. Nous considérons ce chapitre comme l'apport le plus intéressant de notre thèse. Ces travaux ont donné lieu à la présentation d'un poster à une conférence européenne [Laroche *et al.*, 1999a] et à la publication d'un article dans un congrès francophone [Laroche *et al.*, 2000].

- Enfin, notre étude des MDP nous a permis de collaborer aux travaux d'Olivier Aycard sur la localisation d'un robot mobile à l'aide de modules de reconnaissance de balises naturelles. Nous avons décrit cela au chapitre 3 ; ces travaux ont donné lieu à une publication dans une conférence internationale de robotique [Aycard *et al.*, 1998].

Si dans ce mémoire nous avons beaucoup parlé de robotique mobile, ces travaux ne sont pas à cantonner à cette application, bien au contraire. Au sein de l'équipe MAIA, les MDP sont utilisés pour modéliser divers phénomènes : suivis de patients dialysés à domicile, simulation de phénomènes biologiques, etc. Les travaux d'agrégation d'états ou de décomposition présentés dans ce mémoire pourraient avoir un développement dans ces directions : la décomposition de l'environnement à l'aide d'un graphe serait selon nous la technique la plus facilement réutilisable. A partir du moment où il est possible de diviser la tâche à résoudre, il suffit d'identifier les différents facteurs influant sur la valeur des états pour pouvoir valuer le graphe et résoudre le MDP.

Ceci nous amène à décrire les perspectives de notre travail, qui sont présentées au paragraphe suivant.

7.2 Perspectives

Nous avons présenté les principales perspectives de ce travail en fin des différents chapitres. Nous les reprenons ici en y ajoutant des perspectives que nous n'avons pas encore évoquées précédemment.

7.2.1 Techniques d'approximation

Agrégation d'états

Dans ce cadre, nous avons présenté des méthodes qui agrègent les états d'un couloir, dans leur globalité. Mais plus un couloir est long, moins il est réaliste de vouloir le représenter à l'aide d'un très petit nombre d'états, comme nous le faisons. Il serait donc intéressant de procéder par un découpage plus fin de certains couloirs, en fonction de leur longueur. L'idée est de trouver un compromis entre la qualité des politiques générées et

leur temps de calcul, puisque plus le découpage est fin, plus il y a d'états, ce qui influe sur les temps de calcul.

Une deuxième chose qui pourrait améliorer nos résultats est la détection de zones dans lesquelles l'agrégation d'états peut amener à des collisions faisant échouer les missions. Les obstacles formant un cul-de-sac sont typiquement dans ce cas. Il faudrait alors garder la représentation initiale, très fine, et n'agréger que dans les zones où cela est sans danger. La difficulté consiste à détecter ou à apprendre quelles sont les configurations qui se prêtent bien à l'agrégation.

Décomposition

Parmi les différentes perspectives présentées en fin du chapitre 5, nous en rappellerons ici trois, qui nous semblent les plus intéressantes. Les deux premières visent à essayer d'améliorer encore les qualités des politiques obtenues. Une approche hiérarchique pourrait être testée à peu de frais puisqu'elle ne nécessite qu'une adaptation *a priori* mineure des algorithmes déjà écrits. L'approche par analogie demandera plus de travail, puisque nous n'avons pas encore d'idée précise quant aux mécanismes d'adaptation à mettre en œuvre. La troisième perspective serait de tester notre approche de décomposition sur un autre domaine d'applications, afin de vérifier sa généralité.

7.2.2 Supervision de l'exécution

Si nos travaux se sont attachés à développer des techniques rapides d'approximation des politiques, nous ne nous sommes pas assez penché sur l'exécution de ces politiques. De ce fait, la localisation de notre robot est très loin d'être parfaite. Parmi les travaux qu'il reste à effectuer, on pourrait citer l'apprentissage de la fonction d'observation. L'utilisation simple d'une caméra pourrait également aider à la localisation, d'autant plus que des travaux dans ce cadre ont déjà été effectués par Frank Gechter, doctorant de l'équipe MAIA.

7.2.3 Un MDP encore mieux adapté à la robotique

Nous avons présenté dans le chapitre 3 le paramétrage nécessaire à la bonne utilisation d'un MDP. Mais au cours de nos travaux sur l'agrégation d'états, nous nous sommes aperçus que l'adaptation des MDP à la robotique pourrait encore être améliorée. En effet, les méthodes utilisant des MDP choisissent soit une représentation fine de l'environnement, soit une représentation plus grossière, en fonction par exemple des observations faites dans chaque état. Or nous avons vu qu'un couloir peut être traversé de façon sûre à l'aide d'un module de navigation. Celui-ci peut également éviter les obstacles, à condition que leur configuration ne soit pas trop complexe. On pourrait alors adapter la représentation du MDP selon les besoins : les états contenus dans un couloir peuvent être partiellement agrégés « en largeur » (selon la méthode présentée au paragraphe 4.8), alors que les parties de couloirs trop obstruées restent représentées finement. On pourrait alors calculer une politique très proche de l'optimale et très efficace, dans des temps de calcul plus ou moins réduits selon les environnements.

7.2.4 Intégration d'un niveau symbolique

Dans ce mémoire, nous nous sommes principalement intéressés à la planification de trajectoires, sans nous soucier véritablement des buts réels du robot. Or si le robot se déplace d'un point à un autre, c'est pour remplir une mission, par exemple le dépôt du courrier. Pour traiter ce type de mission, il serait nécessaire d'intégrer notre module de planification de trajectoires à un planificateur de tâches plus classique, mais prenant tout de même en compte l'aspect probabiliste des actions. Pour ce faire, nous pourrions nous inspirer de certains des travaux présentés au cours de ce mémoire, par exemple [Boutilier et Dearden, 1994], qui utilisent des données symboliques.

7.2.5 Perspectives à plus long terme

Pendant nos trois années de thèse, nous avons été amenés à lire un certain nombre d'articles plus ou moins proches des domaines de la robotique et des modèles stochastiques. Au fil de ces lectures, nous avons rencontré de nombreuses approches intéressantes. Citons sans ordre de préférence l'utilisation de réseaux Bayésiens, des Diagrammes d'Influence, de l'apprentissage par renforcement, etc. Peut-être un jour aurons-nous l'occasion de nous pencher sur ces travaux, mais ceci est une autre histoire...

Publications personnelles

[1] Pierre Laroche, François Charpillet, and René Schott. Décomposition d'un processus décisionnel de markov à l'aide d'un graphe. In *12ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle (RFIA 2000)*, volume I, pages 515–523, 2000.

[2] Pierre Laroche, François Charpillet, and René Schott. Mobile robotics planning using abstract markov decision processes. In *IEEE International Conference Tools with Artificial Intelligence*, pages 299–306, 1999.

[3] Pierre Laroche, François Charpillet, and René Schott. Decomposition of markov decision processes using directed graphs. In *Poster session of the European Conference on Planning (ECP'99)*, 1999.

[4] Pierre Laroche and François Charpillet. State Aggregation for Solving Markov Decision Problems - An Application to Mobile Robotics. In *IEEE International Conference Tools with Artificial Intelligence*, 1998.

[5] Olivier Aycard, Pierre Laroche, and François Charpillet. Mobile robot localization in dynamic environments using places re cognition. In *proceedings of IEEE International Conference on Robotics and Automation*, pages 3135–3140, 1998.

[6] Pierre Laroche. Modélisation stochastique pour la planification en robotique : tendances actuelles. Technical Report 96R193, CRIN, 1996.

[7] Pierre Laroche. Modélisation stochastique pour la robotique. Rapport de D.E.A., Université Henri Poincaré Nancy 1, 1995.

Bibliographie

[Aycard *et al.*, 1997a] Olivier Aycard, Francois Charpillet, Dominique Fohr et Jean-Francois Mari. Place Learning and Recognition using Hidden Markov Models. Dans *IEEE/ International Conference on Robotics and Intelligent Robot s and Systems*, 1997.

[Aycard *et al.*, 1997b] Olivier Aycard, Francois Charpillet et Jean-Paul Haton. A new approach to design fuzzy controllers for mobile robots navigation. Dans *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1997.

[Aycard *et al.*, 1998] Olivier Aycard, Pierre Laroche et François Charpillet. Mobile robot localization in dynamic environments using places re cognition. Dans *proceedings of IEEE International Conference on Robotics and Automation*, pages 3135–3140, 1998.

[Aycard, 1998] Olivier Aycard. *Architecture de contrôle pour robot mobile en environnement intérieur structuré.* PhD thesis, Université Henri Poincaré Nancy 1, 1998.

[Bellman, 1957] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[Boissonnat *et al.*, 1998] rédacteurs Jean-Daniel Boissonnat, Bernard Faverjon et Jean-Pierre Merlet. *Techniques de la robotique - Tome II : perception et planification.* Hermes, 1998.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas dean et Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[Boutilier et Dearden, 1994] Craig Boutilier et Richard Dearden. Using abstractions for decision-theoretic planning with time constraints. Dans *Proceedings of the National Conference on Artificial Intelligence*, pages 1016–1022, 1994.

[Boutilier et Poole, 1996] Craig Boutilier et David Poole. Computing optimal policies for partially observable decision processes using compact representations. Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[Burgard *et al.*, 1998] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner et S. Thrun. The interactive museum tour-guide robot. Dans *Proceedings of the National Conference on Artificial Intelligence*, 1998.

[Cassandra *et al.*, 1996] Anthony R. Cassandra, Leslie P. Kaelbling et James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. Dans *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, 1996.

[Cassandra *et al.*, 1997] Anthony Cassandra, Michael L. Littman et Nevin L. Zhang. Incremental pruning: A simple, fast, exact algorithm for partially observable markov de-

cision processes. Dans *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*, 1997.

[Cassandra, 1994] Anthony R. Cassandra. Optimal policies for partially observable markov decision processes. Rapport Technique CS94-14, Department of Computer Science, Brown University, Providence, Rhode Island, 1994.

[Cassandra, 1998] Anthony Rocco Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, Department of Computer Science, Brown University, 1998.

[Dean *et al.*, 1993] Thomas Dean, Leslie Kaelbling, Jak Kirman et Ann Nicholson. Planning with deadlines in stochastic domains. Dans *Proceedings of the 11th National Conference on Artificial Intelligence*, 1993.

[Dean *et al.*, 1995] Thomas Dean, Leslie Kaelbling, Jak Kirman et Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1-2):35–74, 1995.

[Dean et Lin, 1995] Thomas Dean et Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. Dans *Proceedings of International Joint Conference on Artificial Intelligence*, 1995.

[Dearden et Boutilier, 1997] Richard Dearden et Craig Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997.

[Dongarra *et al.*, 1993] J. Dongarra, R. Pozo et D. Walker. Lapack++: A design overview of object-oriented extensions for high performance linear algebra. Dans *Proceedings of Supercomputing '93*, pages 162–171. IEEE Computer Society Press, 1993. (voir le site http://math.nist.gov/lapack++/).

[Dutech, 1999] Alain Dutech. *Apprentissage d'environnements : Approches Cognitives et Comportementales.* PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, 1999.

[Fikes et Nilsson, 1971] Richard E. Fikes et Nils J. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Forbes *et al.*, 1995] Jeff Forbes, Tim Huang, Keiji Kanazawa et Stuart Russell. The batmobile: Towards a bayesian automated taxi. Dans *Proceedings of International Joint Conference on Artificial Intelligence*, 1995.

[Fox *et al.*, 1998a] D. Fox, W. Burgard, S. Thrun et A.B. Cremers. A hybrid collision avoidance method for mobile robots. Dans *Proceedings of the IEEE International Conference on Robotics & Automation*, 1998.

[Fox *et al.*, 1998b] D. Fox, W. Burgard, S. Thrun et A.B. Cremers. Position estimation for mobile robots in dynamic environments. Dans *Proc. of the National Conference on Artificial Intelligence*, 1998.

[Haddawy *et al.*, 1995] Peter Haddawy, AnHai Doan et Richard Goodwin. Efficient decision-theoretic planning: Techniques and empirical analysis. Dans *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, 1995.

[Hansen, 1997] Eric A. Hansen. Markov decision processes with observation costs. Rapport Technique CMPSCI TR 97-01, Department of Computer Science, University of Massachussets, Amherst, MA 01003, 1997.

[Hauskrecht *et al.*, 1998] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean et Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. Dans *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998.

[Helwig et Haddawy, 1996] James Helwig et Peter Haddawy. An abstraction-based approach to interleaving planning and execution in partially-observable domains. Dans *Working notes of the 1996 AAAI Fall Symposium on Plan Execution : Problems and Issues*, 1996.

[Howard, 1960] Ronald A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, Massachussets, 1960.

[Jaakkola *et al.*, 1995] Tommi Jaakkola, Satinder P. Singh et Michael I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. Dans *Proceedings of Neural Information Processing Systems 7*, 1995.

[Kaelbling *et al.*, 1996] Leslie Pack Kaelbling, Michael L. Littman et Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, pages 237–277, 1996.

[Koenig et Simmons, 1996a] Sven Koenig et Reid Simmons. Unsupervised Learning of Probabilistic Models for Robot Navigation. Dans *Proceedings of IEEE International Conference on Robotics and Automation*, 1996.

[Koenig et Simmons, 1996b] Sven Koenig et Reid G. Simmons. Passive distance learning for robot navigation. Dans *Proceedings of the thirteenth International Conference on Machine Learning*, pages 266–274, 1996.

[Koenig et Simmons, 1998] Sven Koenig et Reid G. Simmons. Xavier: A robot navigation architecture based on pomdp models. Dans *Artificial Intelligence Based Mobile Robotics*. MIT Press, 1998.

[Kushmerick *et al.*, 1995] Nick Kushmerick, Steve Hanks et Daniel Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2), 1995.

[Laroche *et al.*, 1999a] Pierre Laroche, François Charpillet et René Schott. Decomposition of markov decision processes using directed graphs. Dans *Poster session of the European Conference on Planning (ECP'99)*, 1999.

[Laroche *et al.*, 1999b] Pierre Laroche, François Charpillet et René Schott. Mobile robotics planning using abstract markov decision processes. Dans *IEEE International Conference Tools with Artificial Intelligence*, pages 299–306, 1999.

[Laroche *et al.*, 2000] Pierre Laroche, François Charpillet et René Schott. Décomposition d'un processus décisionnel de markov à l'aide d'un graphe. Dans *12ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle (RFIA 2000)*, volume I, pages 515–523, 2000.

[Laroche et Charpillet, 1998] Pierre Laroche et François Charpillet. State Aggregation for Solving Markov Decision Problems - An Application to Mobile Robotics. Dans *IEEE International Conference Tools with Artificial Intelligence*, 1998.

[Laroche, 1995] Pierre Laroche. Modélisation stochastique pour la robotique. Rapport de D.E.A., Université Henri Poincaré Nancy 1, 1995.

[Littman *et al.*, 1995a] Michael L. Littman, Anthony R. Cassandra et Leslie P. Kaelbling. Learning Policies for Partially Observable Environments: Scaling up. Dans *Proceedings of the 12th International Conference on Machine Learning*, 1995.

[Littman *et al.*, 1995b] Michael L. Littman, Thomas L. Dean et Leslie Pack Kaelbling. On the complexity of solving markov decision problems. *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95), Montreal, Québec, Canada*, 1995.

[Littman, 1994] Michael L. Littman. The witness algorithm: Solving partially observable markov decision processes. Rapport Technique CS-94-40, Brown University, Department of Computer Science, Providence, Rhode Island, 1994.

[Littman, 1996] Michael L. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Brown University, 1996.

[Lovejoy, 1991] William S. Lovejoy. A survey of algorithmic methods for partially observable markov decision processes. *Annals of Operations Research*, 28:47–66, 1991.

[Mahadevan *et al.*, 1998] Sridhar Mahadevan, Georgios Theocharous et Nikfar Khaleeli. Rapid concept learning for mobile robots. *Machine Learning and Autonomous Robots (joint issue)*, 31(5):239–251, 1998.

[Mari *et al.*, 1997] J.-F. Mari, J.-P. Haton et A. Kriouile. Automatic Word Recognition Based on Second-Order Hidden Markov Models. *IEEE Transactions on Speech and Audio Processing*, 5, Janvier 1997.

[Mari, 1996] J.-F. Mari. *Perception de signaux complexes et interaction Homme-Machine.* PhD thesis, Université Henri Poincaré-Nancy I, 1996. Habilitation à diriger des Recherches.

[Marion, 1996] Jean-Luc Marion. *Planification avec Incertitudes.* PhD thesis, Université Paris XI ORSAY, 1996.

[McCallum, 1995] Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State.* PhD thesis, Department of Computer Science, University of Rochester, New York, 1995.

[Monahan, 1982] George E. Monahan. A survey of partially observable markov decision processes: Theory, models and algorithms. *Management Science*, 28(1):1–16, Jan. 1982.

[Nourbakhsh *et al.*, 1995] Illah Nourbakhsh, Rob Powers et Stan Birchfield. Dervish: an office-navigating robot. *AI Magazine*, pages 53–60, Summer 1995.

[Parr et Russell, 1995] Ronald Parr et Stuart Russell. Approximating optimal policies for partially observable stochastic domains. Dans *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1088–1094, 1995.

[Parr, 1998a] Ronald Parr. Flexible decomposition algorithms for weakly coupled markov decision problems. Dans *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998.

[Parr, 1998b] Ronald Parr. *Hierarchical Control and Learning for Markov Decision Processes.* PhD thesis, University of California, Computer Science Division, Berkeley, California, 1998.

[Precup *et al.*, 1998] Doina Precup, Richard S. Sutton et Satinder Singh. Theoretical results on reinforcement learning with temporally abstract options. Dans *ECML-98,*

Proceedings of the 10th European Conference on Machine Learning, pages 382–393, Chemnitz, Germany, 1998. Springer Verlag.

[Precup et Sutton, 1997] Doina Precup et Richard S. Sutton. Multi-time models for temporally abstract planning. Dans *Advances in Neural Information Processing Systems.* MIT Press, 1997.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes.* John Wiley & Sons, New York, 1994.

[Rabiner, 1989] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech recognition. *IEEE Trans. on ASSP*, 77(2):257 – 286, February 1989.

[Shatkay et Kaelbling, 1997] Hagit Shatkay et Leslie Pack Kaelbling. Learning topological maps with weak local odometric information. Dans *Proceedings of International Joint Conference on Artificial Intelligence*, 1997.

[Shatkay, 1998] Hagit Shatkay. *Learning Models for Robot Navigation.* PhD thesis, Department of Computer Science, Brown University, 1998. Technical Report cs-98-11.

[Shatkay, 1999] Hagit Shatkay. Learning hidden markov models with geometrical constraints. Dans *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 1999.

[Simmons et Koenig, 1995] Reid Simmons et Sven Koenig. Probabilistic robot navigation in partially observable environments. Dans *Proceedings of International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995.

[Singh *et al.*, 1994] Satinder P. Singh, Tommi Jaakkola et Michael I. Jordan. Learning without state-estimation in partially observable markovian decision processes. Dans *Proceedings of International Conference on Machine Learning*, 1994.

[Singh, 1994] Satinder P. Singh. *Learning to solve Markovian Decision Processes.* PhD thesis, University of Massassuchets, 1994.

[Smallwood et Sondik, 1973] Richard D. Smallwood et Edward J. Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[Tash, 1994] Jonathan Tash. Issues in constructing and learning abstract decision models. Dans *AAAI Spring Symposium Technical Report on Decision-Theoretic Planning*, 1994.

[Thiébaux *et al.*, 1993] Sylvie Thiébaux, Joachim Hertzberg, William Schoaf et Moti Schneider. A stochastic model of actions and plans for anytime planning under uncertainty. Dans *Proceedings of the 2nd European Workshop on Planning*, 1993.

[Thrun *et al.*, 1998a] S. Thrun, W. Burgard et D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning and Autonomous Robots (joint issue)*, 31(5), 1998.

[Thrun *et al.*, 1998b] S. Thrun, D. Fox et W. Burgard. Probabilistic mapping of an environment by a mobile robot. Dans *Proceedings of the IEEE International Conference on Robotics & Automation*, 1998.

[Tijms, 1986] Henk C. Tijms. *Stochastic Modelling and Analysis, A Computational Approach.* John Wiley & Sons, New York, 1986.

[Washington, 1996] Richard Washington. Incremental markov-model planning. Dans *Proceedings of International Conference on Tools with Artificial Intelligence*, pages 41–47, 1996.

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

[Williams et BairdIII, 1993] Ronald J. Williams et Leemon C. BairdIII. Tight performance bounds on greedy policies based on imperfect value functions. Rapport Technique NU-CCS-93-14, Northeastern University, 1993.

[Zhang et Liu, 1996] Nevin L. Zhang et Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Rapport Technique HKUST-CS96-31, The Hong Kong University of Science and Technology, Department of Computer Science, Clear Water Bay, Kowloon, Hong Kong, 1996.

université HENRI POINCARÉ
NANCY 1

FACULTÉ DES SCIENCES

Monsieur **LAROCHE Pierre**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 7 février 2000 n° 364

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY 1 — LE PRÉSIDENT

Cl. BURLET

Résumé

Mots-clés: Planification, Processus Décisionnel de Markov, Agrégation d'états, Décomposition, Robotique.

Un robot mobile évoluant dans un environnement dynamique doit faire face à de nombreuses incertitudes. Les actions qu'il exécute n'ont pas toujours l'effet escompté, et ses capteurs lui renvoient des données souvent bruitées. Les Processus Décisionnels de Markov (MDP), du fait de leur adaptation à la prise en compte d'incertitudes, sont étudiés depuis quelques années dans la communauté Intelligence Artificielle. Ceux-ci permettent le calcul d'un plan plus robuste que les méthodes de planification classiques, puisque calculé en prenant en compte les diverses conséquences probables des actions. Dans le cadre de la robotique mobile, ce modèle permet également de superviser l'exécution des missions, en utilisant des fonctions probabilistes pour aider le robot à se localiser. Mais les algorithmes classiques de résolution de MDP sont complexes, ce qui les rend difficilement adaptables à des environnements de grande taille, tels que ceux nécessités dans le cadre de la robotique.

En conséquence, beaucoup de travaux s'intéressent aux techniques d'approximation, qui permettent d'obtenir des plans sous-optimaux en un temps de calcul réduit. C'est dans ce cadre que s'inscrit cette thèse. Après avoir montré comment nous paramétrons un MDP pour l'appliquer à la robotique mobile, nous présentons deux techniques d'approximation. La première utilise l'agrégation d'états, chaque couloir de l'environnement donnant lieu à un ou plusieurs états abstraits. La seconde est fondée sur la décomposition d'environnements, lesquels sont représentés à l'aide d'un graphe valué permettant d'approcher heuristiquement les valeurs des états. Ces deux méthodes permettent de réduire considérablement les temps de calcul, et donnent des plans très proches de l'optimal.