Système délibératif d'un robot autonome : planification probabiliste hiérarchique basée sur des motivations et prise en compte de ressources

Raphaël Gottstein

THÈSE DE DOCTORAT DE
L'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Raphaël GOTTSTEIN

Pour obtenir le grade de
DOCTEUR de L'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Système délibératif d'un robot autonome : planification probabiliste hiérarchique basée sur des motivations et prise en compte de ressources

soutenue le 13 juillet 2017

devant le jury composé de :

M. Raja Chatila	Directeur de thèse
M. Rachid Alami	Rapporteur
M. Abdel-Illah Mouaddib	Rapporteur
M. François Charpillet	Examinateur
Mme. Catherine Pelachaud	Examinateur
Mme. Catherine Tessier	Examinateur

Remerciements

Je n'ai jamais su dire correctement "merci", mais je fais des efforts pour m'améliorer. Un peu plus de trois ans après avoir commencé ce long et parfois douloureux travail, je voudrais remercier ceux qui ont étés là. J'en oublie forcément, désolé pour eux.

Je voudrais tout d'abord (c'est l'usage, mais c'est sincère) remercier mon directeur de thèse, Raja Chatila, qui, malgré la charge de travail qu'il s'inflige, m'a toujours accordé une attention bienveillante. Malgré la fatigue, le stress et les déconvenues, c'était un plaisir.

Je remercie bien sûr Rachid Alami et Abdel-Illah Mouaddib qui ont accepté d'être rapporteurs et de lire ce documents et toutes les fautes qu'il contient. J'en profite pour remercier Abdel-Illah avec qui j'ai par le passé déjà travaillé pendant mon master, merci pour son soutien qui m'a permis de faire cette thèse. Merci également à François Charpillet, Catherine Pelachaud et Catherine Tessier d'avoir accepté d'être examinateurs. J'espère que vous ce travail vous plaira.

Merci à l'équipe du projet RoboErgoSum, Mehdi Khamassi, Benoît Girard, Aurélie Clodic, Erwan Renaudo, Scarlett Fres, Ricardo Omar Chavez-Garcia, Pierre Luce-Vayrac, Rachid et Raja. Merci pour ces réunions intrigantes avec une bonne ambiance. Désolé de ne pas avoir été là pour les dernières réunions.

Je voudrai remercier l'équipe administrative et technique de l'ISIR, qui s'est toujours bien occupé de moi avec gentillesse, même pour des histoires de portes qui grincent.

Merci au bureau J03 qui, entre ses quatre murs m'a abrité moi ainsi que Arthur Bouton, Alexandra Pimenta Dos Santos, Ziad Zamzami, Carlos Maestre et Oscar De La Cruz, mais également quelques occupants passagers, Mihai Andries (qui m'a bien aidé) et Omar Islas Ramirez. Je voudrai aussi remercier plein de gens, des habitants et habitantes du laboratoire, mais comme je suis sûr et certain d'en oublier, je n'essaie pas d'énumérer vos noms. Merci à vous tous pour ces moments passés pendant ces trois ans.

Je voudrai également remercier tous les intervenants, intervenantes, doctorants et doctorantes que j'ai rencontré quelques heures au détour des formations auxquelles j'ai participé. Vous avez été une bouffée d'air très bienvenue pendant une période difficile.

Merci enfin à mes proches, qui m'ont soutenu, sans nécessairement comprendre les difficultés que cet exercice comprend. Merci mon papa et ma maman qui m'ont permis de voir plus loin, en étant derrière moi et sans trop me presser. Merci aux frangins, Rodolphe et Cyprien, qui ont toujours su s'intéresser à ce que je faisais et pour tous les bons moments passés à penser à autre chose. Merci aux parents de ma chérie, Patricia et Didier, qui m'ont poussé comme ils poussent leurs propres enfants.

Merci à tous les amis de Caen que je n'ai revu que trop peu de fois depuis le début de cette thèse. J'ai hâte de pouvoir vous revoir tous une fois cette thèse finie.

Merci enfin à ma chérie, Clara. La vie aurait été insupportable sans toi.

Je ne saurai jamais si cette thèse aura "valu le coup", si ce travail servira à quelque chose un jour ou est destiné à se perdre dans les méandres de la recherche. Malgré toutes les difficultés, les doutes, les fois où j'ai pensé abandonner, je suis quand même assez fier de ces trois années.

Résumé

Les travaux menés dans le domaine des sciences de la décision ont depuis longtemps permis d'apporter des réponses efficaces permettant de résoudre des problèmes liés à l'incertitude, à la temporalité des actions ou à la coopération entre agents. Toutes ces avancées ont permis d'avancer vers la capacité pour un agent artificiel à être autonome dans son environnement. Cependant, la capacité pour un agent à être autonome dans le choix de ses objectifs reste un problème difficile à traiter. Cette faculté est portant essentielle dans les cas où aucun ingénieur ne peut choisir ou aiguiller un robot dans le choix des tâches à accomplir. Comment permettre à un robot n'ayant pas la possibilité d'être assisté de décider lorsqu'il doit résoudre plusieurs objectifs dans un environnement incertain? C'est le problème que nous avons posé : permettre à un robot de planifier ses actions pour de multiples objectifs complexes possiblement contradictoires, dans un environnement probabiliste, avec la présence de ressources (énergie, temps), et pour des problèmes sans limite de temps explicite. La notion de ressources s'impose dans ce contexte où la temporalité des actions, qui n'ont pas toutes la même durée, est importante, de même que la nécessité de gérer l'énergie du robot dans le temps. Dans ce manuscrit, nous proposons une solution à ce problème en apportant une nouvelle méthode permettant de définir des objectifs complexes pour un robot sous la forme de *motivations* et en présentant un processus de planification hiérarchique permettant de planifier la progression des objectifs posés.

La première contribution de ce travail est la présentation des *motivations*. Pour permettre de définir des objectifs bouclants et contenant des enchaînements de tâches, nous choisissons d'élaborer des automates qui décrivent la manière dont évolue un objectif en fonction de l'évolution de l'état du monde. Une motivation définit plusieurs états, à partir desquels la réalisation de certaines actions ou le fait d'atteindre certains états est significant, faisant changer d'état la motivation. Ce sont également les motivations qui définissent comment les récompenses du systèmes sont obtenues, lors du passage d'un état à un autre.

Le seconde partie de ce travail consiste en la création d'un processus hiérarchique permettant de traiter les motivations définies. Pour cela, l'objectif est de calculer des politiques optimales permettant faire évoluer l'état des motivations. Ces politiques sont traitées afin de créer des macro-actions, correspondant au modèle de leur exécution, qui sont utilisées dans un second temps lors d'une planification locale. Pour faire progresser la résolution d'un objectif, nous proposons de calculer des politiques dont le rôle est de déclencher un changement d'état d'une motivation. Dans un environnement complexe, il est fortement probable que la résolution d'une unique tâche n'exploite pas la totalité du modèle de l'environnement. Nous utilisons donc un processus de décision markovien (MDP) permettant de calculer une politique sur une sous-définition du modèle ne contenant que les éléments nécessaires à la réalisation de la tâche, ce qui nous permet de limiter fortement les coûts des calculs. Ensuite, pour pouvoir prédire le comportement d'une politique, nous présentons plusieurs méthodes d'analyse utilisant de vastes systèmes d'équations linéaires. Ces calculs nous aident à construire un modèle de l'exécution des politiques (état finaux, ressources consommées), que nous appelons des *macro-actions*. Celle-ci nous permettent de connaître la probabilité d'un changement d'état d'une motivation et permet de connaître les futures tâches à résoudre. Une fois les macro-actions définies, nous cherchons une solution au problème. La taille du problème et la présence de ressources, représentées par des valeurs numériques, rendent les méthodes de planification

complète inutilisables. Nous choisissons de développer un arbre partant de la situation courante et décrivant l'ensemble des état futurs possibles, mais omettant les scénarios trop improbables. L'arbre possède un horizon exprimé grâce à la ressource de temps, et permettant de comparer des scénarios ayant une durée équivalente (plutôt qu'un nombre de macro-actions équivalent). L'arbre est ensuite élagué pour sélectionner les macro-actions qui promettent d'obtenir le plus de récompense. Cela forme notre solution au problème posé, un *agenda de macro-actions*.

Pour finir, nous proposons d'inclure ce système de planification au sein d'une architecture délibérative, composée de trois modules principaux : un module opérationnel (qui calcule et analyse les politiques), un module délibératif (qui calcule les agendas de macro-actions), et un module superviseur (qui contrôle l'exécution des agendas). Cette structure nous permet de définir un cycle délibération / exécution, et ainsi de rendre le calcul d'une solution possible dans la durée. L'architecture délibérative permet de contrôler l'exécution des agendas de macro-actions produits et de demander la production de nouveaux agendas lorsque cela est nécessaire. Cette architecture a été implémentée et testée en simulation sur un problème complexe et de grande taille, représentatif de la problématique traitée. Nous avons pu observer que l'architecture fonctionne rapidement et avec peu de ressources informatiques, et qu'elle produit les comportements logiques attendus, en termes d'optimisation des gains et de minimisation des risques.

Sommaire

Liste des figures

Liste des tables

Liste des termes

MDP Processus de Décision Markovien (Markovian Decision Process en anglais). Modèle de décision stochastique dans lequel les actions produisent un résultat incertain, donné par une table de transition.

ws État du monde (world state en anglais). Habituellement abrégé *s*, l'état du monde et du robot. Abrégé ici *ws* pour éviter les confusions avec les motivation states et global states.

HTN Hierarchical Task Network. Dans le domaine de la décision déterministe, méthode permettant de créer des macro-actions. Une macro-action est un plan résumé sous la forme d'un ensemble de préconditions et d'effet, et devient donc utilisable lors d'une planification au même titre que les action normales.

motivation Automate utilisé pour représenter un objectif. Les états (ms) représentent différents statuts de l'objectif et les transitions (rmt) sont les modalités de passage d'un état à un autre.

ms État de la motivation (motivation state en anglais). État de l'automate d'une motivation.

msv Vecteur des états des motivations (motivation state vector en anglais). Un vecteur contenant les états de chaque motivations.

rmt Transition motivationnelle récompensée (rewarded motivation transition en anglais). Définit le passage de l'état de la motivation ms à ms' lorsque la transition finale ft est exécutée et la récompense (positive ou non) obtenue lors de son passage. Une rmt s'écrit $(ms,ft,ms'),r$.

gt Transition but (goal transition en anglais). Définit une transition (ws,a,ws') comme but, c'est-à-dire une transition visée qui, lorsqu'elle apparaît, entraîne la fin de l'exécution de la politique en cours.

rwt Transition du monde récompensée (rewarded world transition en anglais). Extraite de la d'une rmt, elle contient une ft et une récompense, et est notée $(ws, a, ws'), r$. Utilisée par un MDP customisé pour créer π_{rwt}.

R_{sys} Récompense accumulée par le système décisionnel. Correspond à la somme des récompenses accumulée par chacune des motivations.

PSO Probabilistic STRIPS Operator. Description d'une action afin d'être utilisée dans un modèle de décision probabiliste. Il décline le définition des opérateurs de STRIPS, un langage de description de problème déterministe.

WM Définition d'un modèle (ou d'un sous-modèle) du monde avec PSO. Défini comme $WM = (V, O)$, avec V l'ensemble de variables d'état et O l'ensemble des opérateurs.

π_{rmt}^{msv} Politique visant rmt depuis msv, également notée π_{RWT}. Politique visant la résolution de la rwt issue de rmt et qui, une fois exécutée, se termine. Elle sert à provoquer l'exécution d'une rmt et est utilisée par le module de délibération comme une action.

$Pr(\boldsymbol{rwt}|\boldsymbol{ws}, \pi)$ Probabilité de finir l'exécution de la politique π par l'exécution de rwt, en commençant depuis ws.

$\boldsymbol{NbAction}(rwt, ws, \pi)$ Nombre moyen d'action utilisées pour finir l'exécution de la politique π par l'exécution de rwt, en commençant depuis ws.

gs État global (global state en anglais). Couple composé des états ws et msv. Définit l'état du système.

ma Macro action. Modèle d'exécution d'une politique π_{rmt}^{msv}.

$T(gs, ma, gs')$ Fonction de transition globale. Donne la probabilité de la transition globale (gs, ma, gs').

π^r méta-politique pour les états remarquables. Donne la macro action ma optimale à exécuter depuis l'ensemble des états remarquables (ensemble des états globaux atteints à la fin des macro actions).

π^g méta-politique donnant la macro action ma pour l'ensemble des états globaux gs.

$C_r(ws, a, ws')$ Fonction de coût. Donne pour chaque transition (ws, a, ws') le coût pour la ressource r.

$C_r(rwt, ws, \pi)$ Fonction de coût. Donne le coût moyen pour la ressource r de l'exécution de la politique π, débutée depuis l'état ws et finissant par l'exécution de rwt.

Noter que pour des raisons pratique, des termes anglais seront parfois préférés dans ce document.

Chapitre 1

Introduction

Contents

1.1 Contexte

Depuis une centaine d'années, les sciences de la décision donnent des outils variés pour résoudre des problèmes d'intelligence artificielle ou de robotique. Les études ont théorisé la décision pour de nombreux domaines, pour les problèmes liés à l'incertitude, à la temporalité des actions, ou au nombre d'acteurs du problème (alliés ou adversaires).

Autonomie dans le choix des objectifs. Si les méthodes de décision sont de plus en plus riches, certains problèmes restent à nos jours difficiles à appréhender. C'est le cas de l'autonomie d'un agent dans le choix de ses objectifs, c'est-à-dire la capacité pour un agent à choisir quels objectifs satisfaire et dans quel ordre, et ce, sans aide extérieure. Cette capacité est effet fondamentale dans le cadre des robots autonomes pour lesquels aucun ingénieur ne sera présent pour les aiguiller dans leur mission. Cette limitation peut provenir d'une difficulté ou impossibilité de communiquer (les véhicules autonomes sur Mars, par exemple), ou d'un choix (un robot d'assistance domestique). Il s'agit toujours de cas où un robot a un ensemble de tâches, parfois contradictoires, à accomplir dans un environnement complexe et imprévisible.

Être autonome, c'est choisir comment investir ses possibilités, son temps (un robot ne peut être qu'à un endroit à la fois) et ses ressources (énergie, par exemple). Cela implique de savoir si une tâche peut être réalisée, et ce qu'elle implique pour la réalisation des autres tâches.

RoboErgoSum. Le projet RoboErgoSum[1], dans lequel s'inscrit ce travail, a pour objectifs d'explorer les outils nécessaires à un robot pour lui permettre de comprendre son environnement, de prendre des initiatives , de raisonner, sans aide extérieure et d'apprendre de ses expériences et de savoir ce qu'il a appris. Pour atteindre l'ensemble de ces capacités, nous pensons que le robot doit développer une conscience de soi (*self-awareness* en anglais).

Dans ce projet, la notion de *motivation* a été proposé pour représenter les objectifs internes et structurés du robot. Ce travail de thèse a pour objectif d'explorer ces motivations et de créer un système décisionnel permettant de les satisfaire. Pour cela le système donnera des objectifs permanents ne pouvant être définitivement résolus. Ces motivations pourront être partiellement satisfaites par la réalisation de certaines actions ou en atteignant certains états. Pour satisfaire ces objectifs, nous développerons un système capable de calculer des plans pour satisfaire les objectifs d'une part et de planifier l'ordre dans lequel exécuter ces plans d'autre part.

1.2 Problématique générale

Notre objectif est de trouver une méthode permettant de résoudre un problème constitué d'objectifs multiples et possiblement contradictoires en un temps raisonnable. De plus, et pour répondre aux besoins de traiter des scénarios réalistes, le problème devra tenir compte de l'incertitude des actions et inclure au moins deux ressources : le temps et l'énergie sur lesquelles nous nous limiterons. Il s'agira pour finir de traiter des problèmes sans fin explicite, afin de répondre à la problématique de l'autonomie complète et continue.

[1] Projet ANR ANR-12-CORD-0030, http://roboergosum.isir.upmc.fr/

Caractéristiques. L'ensemble des caractéristiques que nous devons traiter conjointement est donc :

- **Décision probabiliste** : pour tenir compte de l'inexactitude des modèles utilisés, le modèle du problème choisi sera donc probabiliste.
- **Sans limite de temps explicite** : notre problème ne peut être résolu à l'aide d'un plan de longueur fini. Nous devons trouver des solutions pour traiter le problème sur toute son étendue.
- **Avec des objectif structurés** : nous devons développer la notion de *motivation*, c'est-à-dire des objectifs complexes et valables dans le temps. Nous devons définir une structure permettant de définir des objectifs et être capable de tenir compte de cette structure lors de la recherche d'une solution.
- **Prise en compte de ressources** : afin de pouvoir représenter des problèmes en autonomie totale, nous devons être en mesure de prendre en compte des ressources comme le temps et l'énergie. En conséquence, le problème de décision comportera également une gestion du niveau des ressources.

1.3 Approche proposée

Pour résoudre le problème posé, nous proposons une approche en trois temps, permettant de partir de la base du problème. Nous commençons par la définition des objectifs à l'aide des *motivations*, pour aller vers la planification, permettant de générer des politiques utilisables pour satisfaire en partie un objectif puis de calculer un plan conditionnel utilisant les politiques précédemment calculées. Enfin, nous proposerons d'intégrer ce processus de planification dans une architecture délibérative, qui permettra de résoudre le problème posé sur la longueur, en alternant des phases de délibération et des phases d'exécution supervisée.

Définition des motivations. Nous proposons, pour définir les objectifs, de créer une structure de donnée utilisant des machines à nombre d'états fini, et que nous nommerons *motivations*. Cette structure permettra de définir des états motivationnels permettant de caractériser des situations particulières au sein d'un objectif. Cela pourra être simplement deux états, *activé* / *désactivé*, ou plusieurs états se succédant, permettant de différencier des étapes au sein d'un processus et autorisant également la création d'embranchements. Les motivations permettront également de définir des transitions entre états motivationnels, conditionnées par le fait d'atteindre un état, de réaliser une action ou d'atteindre un niveau de ressource. Ces transitions définiront des tâches que le robot pourra accomplir pour faire progresser les objectifs. Ces transitions comporteront une valeur de récompense, permettant de quantifier leur aspect positif ou négatif. Cette récompense aura le même rôle que dans les MDP classiques, et leur maximisation sera l'objectif de l'architecture finale.

Planification en deux temps. Pour pouvoir traiter des problèmes complexes et de grande taille, nous allons développer une méthode hiérarchique pour calculer une solution en un temps raisonnable.

En effet, avec les processus markoviens, lorsque le nombre et la complexité des objectifs augmente, le temps et l'espace requis pour les calculs explose. Les *motivations* proposées plus tôt nous permettront alors de décomposer le modèle markovien classique en deux : un **modèle opérationnel** et un **modèle intentionnel**. Le modèle intentionnel sera constituée des motivations, tandis que le modèle opérationnel sera constitué d'un modèle MDP classique auquel nous avons retiré la fonction de récompense et pour lequel seul l'environnement réel du robot et ses capacités d'action sont représentées. À partir de ces deux modèles séparés, nous allons mettre en place un système de planification en deux temps. L'objectif est de pouvoir utiliser des politiques générées pour déclencher des transitions des motivations (permettant donc d'obtenir les récompenses associées) afin de maximiser la récompense obtenue par les motivations.

Pour commencer, nous allons utiliser le modèle opérationnel pour calculer, une à une, des politiques permettant aux motivations de déclencher une transition entre deux états motivationnels. Celles-ci étant conditionnées par l'exécution d'une action par exemple, nous allons calculer des politiques stoppantes pour l'exécution de ces actions. Grâce à un MDP modifié, nous pouvons calculer la politique optimale pour exécuter une action donnée. De plus, nous proposons un système permettant d'utiliser des versions réduites du modèle opérationnel en fonction de l'action que nous cherchons à réaliser. Cette réduction est possible en définissant un modèle factorisé à partir des actions nécessaires pour la réalisation de l'objectif : une tâche ne requérant que des actions de navigation par exemple ne modifiera jamais l'état des variables de l'environnement qui ne sont pas relatives aux déplacements du robot. Ainsi, chaque politique pourra être calculée sur un modèle réduit propre à la tâche que nous souhaitons accomplir. Ces politiques seront toutes construites avant de passer à la seconde phase de la planification.

Pour finir, nous utiliserons les politiques calculées pour générer des **macro-actions**, c'est-à-dire le modèle d'exécution de ces politiques. Les méthodes que nous proposons dans ce travail permettent d'avoir une forte capacité de prédiction sur leur exécution. Ainsi, nous pourrons utiliser les macro-actions pour résoudre le problème dans sa globalité. L'objectif est ainsi de générer une multitude de politiques partielles, ce qui est est moins coûteux que de résoudre une seule fois le problème complet. Nous créons un modèle global ayant pour pour seules actions les macro-actions générées. Ce modèle étant trop grand pour être traité avec des méthodes complètes, nous proposons d'utiliser une planification locale. À l'aide des macro-actions, qui peuvent représenter l'exécution de dizaines ou de centaines d'actions et qui permettent d'accomplir une transition entre deux états motivationnels, nous générons les combinaisons de macro-actions qu'il est possible de réaliser pour un horizon donné. En rapportant les récompenses associées aux transitions des motivations et grâce au modèle des macro-actions, nous pouvons ensuite sélectionner les macro-actions maximisant l'espérance de la récompense obtenue par les motivations pour l'exécution de ces macro-actions. Nous obtenons ainsi un **agenda de macro-actions** qui sera à exécuter.

Conception d'une architecture délibérative. La taille et la complexité, due aux multiples objectifs concurrents et à la présence de ressources, nous oblige a envisager une planification locale plutôt que complète. Le problème posé considérant un problème sans limite de temps explicite, nous proposons d'utiliser notre méthode de planification au sein d'une architecture délibérative. Cette architecture aura pour premier rôle de pouvoir contrôler l'exécution de l'agenda de macro-actions retourné par le processus de délibération. En effet, le modèle choisi étant stochastique, le nombre d'actions pour ter-

miner la macro-action en cours est aléatoire, de même que la consommation en ressources. Il est donc nécessaire de superviser l'exécution de l'agenda des macro-actions. Le second rôle de l'architecture est d'introduire un cycle délibération / exécution, afin d'enchaîner les phases de planification et d'exécution des agendas calculés.

1.4 Plan du mémoire

Nous avons choisi de diviser la présentation de notre travail en six parties, en partant de la présentation de l'état de l'art concernant le problème que nous avons choisi de traiter, puis en développant les solutions pour répondre à notre problématique sur trois chapitres. Les deux derniers chapitre concerneront les résultats expérimentaux que nous avons obtenus ainsi que la conclusion de ce mémoire.

Dans le **Chapitre 2**, intitulé "État de l'art", nous passerons en revue différents travaux ayant pour thème la prise de décision. À notre connaissance, aucun travail n'a été effectué sur l'ensemble des caractéristiques du problème posé. Ainsi, nous avons décidé de présenter des travaux apportant des réponses partielles et regroupés selon trois axes : les travaux traitant de décision déterministe, les travaux ayant une approche probabiliste et enfin les architectures délibératives.

Le **Chapitre 3**, intitulé "Structurer de multiples objectifs" est consacré à la définition de nos objectifs structurés, les **motivations**. Nous commencerons ce chapitre en présentant un problème exemple correspondant à la problématique posée qui servira de base aux exemples futurs et que nous réutiliserons dans les chapitre 4 et 5. Nous verrons ensuite deux méthodes pour modéliser des objectifs dans les MDP. Les limitations de ces deux méthodes nous permettent enfin d'introduire les motivations, des objectifs structurés à partir d'automates. Nous constituerons à cette occasion les motivations correspondant aux objectifs présentés pour l'exemple récurrent.

Pour simplifier l'exposé de notre solution, nous avons choisi de présenter tout d'abord une méthode au problème en ignorant la notion de ressource. Le **Chapitre 4**, intitulé "Architecture pour une solution hiérarchique et factorisée", présentera donc la plus grande partie du travail que nous avons développé, mais sans tenir compte de la problématique des ressources. Ce chapitre présentera tout d'abord l'architecture que nous avons conçue et sa division en trois modules principaux : intentionnel, opérationnel et délibératif. Ces trois modules seront ensuite développés séparément et dans cet ordre. Nous présenterons donc le module intentionnel, gérant les motivations présentes dans le problème. Le module opérationnel, dont la fonction est de retourner des politiques demandées par le module délibératif, ainsi que des fonctions afin de prédire le comportement de ces politiques. La présentation de ce module commencera par la présentation du formalisme que nous avons choisi pour représenter le modèle du monde, les PSO (*Probabilistic STRIPS Operators*), et les capacités de ce modèle à générer des sous-modèles, définis à partir d'un sous-ensemble de variables d'états et d'actions. Nous utiliserons l'exemple récurrent pour illustrer le fonctionnement de ce formalisme. Nous présenterons ensuite la méthode que nous avons choisie afin de calculer des politiques stoppantes, c'est-à-dire permettant de façon optimale de viser la réalisation (une unique fois) d'une action parmi un ensemble d'actions visé. Ensuite, nous développerons les méthodes permettant de prédire le comportement des politiques calculées précédemment. Ces méthodes nous permettront de connaître, pour une politique et à partir d'un état donné la probabilité de finir par une action but donnée ainsi que le nombre moyen d'action effectuées pour l'exécuter. Enfin, nous développerons un module délibératif permettant de résoudre le problème sans ressources, en proposant

de calculer une méta-politique. Cette partie sera tout d'abord l'occasion de présenter comment le module délibératif sélectionne les buts à envoyer au module opérationnel. Nous présentons une méthode permettant d'obtenir des buts garantissant qu'aucune transition entre deux états motivationnels ne peut se déclencher en dehors des actions finales des politiques, ce qui est une condition pour assurer la validité du modèle des macro-actions. Nous proposerons enfin un modèle global rassemblant les modèles intentionnels et opérationnel et ayant pour actions les macro-actions, dont nous présenterons la formation. Pour finir, nous proposerons deux méthodes pour le calcul d'une méta-politique permettant de maximiser l'espérance de la récompense obtenue par les motivations, la première permettant de diminuer considérablement la taille de l'espace d'état, la seconde, plus optimale mais plus longue à calculer, donnant la meilleure macro-action à utiliser pour tous les états.

Le **Chapitre 5**, intitulé "Architecture pour une solution hiérarchique et factorisée avec ressources", reprendra le Chapitre 4 en ajoutant les éléments liés à la problématique des ressources. Ce chapitre débutera par une analyse de la notion de ressource et notre choix pour pouvoir les modéliser. Ces changements nous permettront d'introduire l'architecture finale que nous proposons. Nous présenterons les changements que nous devons opérer dans la constitution des macro-actions et en particulier la nécessité d'estimer la consommation en ressources d'une politique exécutée à partir d'un état donné. Nous proposeront enfin une méthode de planification locale permettant d'apporter une solution au problème posé avec ressources. En effet, la présence des ressources créée une infinité d'états possibles et nous empêche donc d'utiliser le calcul d'une méta-politique. Nous présenterons alors une méthode permettant de développer un arbre partant de la situation courante et donnant tous les futurs possibles et leur probabilité en fonction des combinaisons de macro-actions choisies. Cet arbre sera ensuite élagué pour obtenir un plan conditionnel de macro-actions appelé **agenda de macro-actions**, maximisant la récompense obtenue par les motivations. Pour finir, nous étudierons comment ce calcul peut s'insérer dans un cycle délibération / exécution.

Les résultats expérimentaux seront exposés dans le **Chapitre 6**. Nous commencerons ce chapitre en présentant le problème que nous avons choisi d'évaluer (proche de l'exemple récurrent du chapitre 3, mais plus grand et plus complexe), ainsi que la configuration matérielle que nous avons utilisée. En l'absence d'autres travaux ayant la capacité de traiter notre problème, nous avons choisi de tester les performances de notre processus de délibération. Nous avons pour cela testé plusieurs configurations et avons analysé comment le temps de calcul moyen pour le calcul d'un agenda et la récompense totale obtenue de la part des motivations pouvaient varier lors des expérimentations. Les résultats seront ensuite discutés, sur les tests spécifiquement, et sur les performances dues à la structuration que nous avons choisi de mettre en place.

Enfin, le chapitre 7 conclura en résumant notre contribution, ses avantages et ses limitations et en identifiant des pistes pour des recherches futures.

Chapitre 2

État de l'art

Contents

2.1 Introduction

Le problème auquel nous nous intéressons n'est pas pas directement traité dans la littérature, mais beaucoup de travaux y répondent en partie. Avant de nous intéresser à ces travaux, nous définissons avec précision les caractéristiques de notre problème.

Nous posons un problème de décision :

- **Décision probabiliste** : pour tenir compte de l'incertitude liée à l'environnement et aux inexactitudes inévitables des modèles, le paradigme probabiliste permet de traiter et de modéliser l'incertitude de l'évolution du problème et en particulier des résultats des actions effectuées par l'agent.

- **Sans limite de temps explicite** : les problèmes auxquels nous nous adressons n'ont pas de fin. Nous ne recherchons pas la résolution définitive d'un ensemble d'objectifs uniques, mais plutôt à obtenir un comportement satisfaisant un ensemble d'objectifs permanents (conserver un niveau minimal d'énergie, garder un endroit propre, récolter des roches lunaires, etc.).

- **Avec des objectif structurés** : afin de définir des objectifs complexes et valables dans le temps, les buts exprimés simplement comme un état à atteindre ou une action à réaliser ne sont pas suffisants. Les objectifs de notre problème sont structurés. Ils sont **hiérarchisés**, afin de définir un objectif comme la réalisation successive et logique d'un ensemble de tâches. Ils peuvent de plus être bouclants, c'est-à-dire que l'objectif ne peut être fini, chaque résolution de tâche aboutissant toujours à la nécessité de réaliser une nouvelle tâche.

- **Autonome dans le choix de ses objectifs** : afin de considérer un environnement riche en interaction et puisque notre exemple contient des objectifs sans fin, nous posons à l'agent des **objectifs** éventuellement **contradictoires**. Le robot n'a pas la possibilité de réaliser l'ensemble des buts, il doit choisir tout au long de la résolution du problème les objectifs à satisfaire.

- **Prise en compte de ressources** : pour un robot autonome dans son environnement, et pour faire face à des scénarios réalistes, les ressources comme le **temps** et l'**énergie** doivent être pris en compte. Au-delà d'une simple utilisation comme une date butoir signifiant la fin d'une expérimentation ou comme un paramètre d'optimisation, nous envisageons le temps afin de déclencher des événements tout au long de l'exécution de notre problème. Le temps nous permet de définir des tâches valides pendant une période de temps limitée ou des tâches périodiques. Le niveau de la batterie représente l'énergie disponible pour le robot pour effectuer des actions dans son environnement. La batterie n'est pas entièrement limitante et peut être rechargée. Elle induit de fait une contradiction entre les objectifs nécessitant une dépense de l'énergie pour être réalisés et l'objectif sous-jacent de conserver un niveau d'énergie raisonnable.

À notre connaissance, aucun travail n'a traiter l'ensemble de ces caractéristiques à la fois. Cependant, la littérature dispose d'une profusion de réponses partielles, parfois sur un point précis, parfois sur un ensemble d'éléments. Pour explorer ces travaux, nous avons choisi trois axes. Les deux premiers correspondent à deux grande catégories de la décision : la décision déterministe et la décision probabiliste. Pour expliquer le choix de ces deux

axes, nous passons en revue les forces et faiblesses de ces deux paradigmes des systèmes de décision et l'impact sur la capacité à traiter notre problème. Nous nous intéressons enfin aux architectures délibératives, ces structures assemblant une multitude de processus pour résoudre des problèmes complexes, de robotique particulièrement.

Décision déterministe. La décision déterministe se caractérise par le fait que les effets produits par les actions sont toujours les mêmes. Donc si l'action a est exécutée à partir de l'état $s1$, l'état suivant sera toujours le même état $s2$. Un modèle déterministe s'écrit à l'aide de la logique des prédicats, sa solution produit un plan, c'est-à-dire une suite d'actions à réaliser pour aller de la situation initiale à la situation objectif. En pratique cependant, il n'est pas rare que l'exécution d'une action échoue. En effet, l'aspect déterministe ne se vérifie souvent pas lors d'un passage à une expérimentation réelle. Dans ce cas, des mécanismes de réparation de plans interviennent lors de l'exécution lorsque celle-ci échoue.

Les modèles déterministes peuvent modéliser une grande variété de problèmes, et de nombreux modèles permettent de prendre en compte des ressources. De plus, la question des objectifs structurés, d'objectifs contradictoires et de problèmes sans fin a été abordée dans la littérature.

Décision probabiliste. Ce domaine de la décision fait partie d'une plus grande catégorie : la décision stochastique, où les effets des actions sont variables. La décision probabiliste se caractérise par le fait que les effets des actions sont donnés par des règles probabilistes. En d'autres termes, l'action a exécutée à partir de $s1$ pourra par exemple mener à $s2$ dans 30% des cas et dans $s3$ dans 70% des cas. Un modèle probabiliste simple se définit à l'aide d'un ensemble d'état, un ensemble d'actions, ainsi qu'une fonction permettant d'attribuer à chaque couple état/action, l'ensemble des états dans lesquels il est possible d'arriver et la probabilité d'y arriver. Ce modèle définit également une fonction de récompense, permettant d'attribuer une valeur numérique à un état (ou à un couple état/action selon le modèle). Cette valeur définit la quantité de récompense obtenue lorsque l'état est atteint. Contrairement à la planification déterministe qui retourne un plan, la planification probabiliste produit une politique, c'est-à-dire la meilleure action à exécuter (en probabilité) pour que l'espérance des gains de récompense soit maximale.

La littérature sur la décision probabiliste est importante et les problèmes qu'elle traite sont variés. Cependant, la gestion de multiples objectifs, de ressources ou simplement les problèmes de grande taille sont un défi pour ces systèmes, pour lesquels l'explosion des espaces de recherche est rapide. Pour compenser cette explosion, des méthodes hiérarchiques sont développées afin de structurer les problèmes et de les résoudre sous-problèmes par sous-problèmes.

Architectures délibératives. Pour finir, nous nous intéressons aux architectures délibératives, qui sont le moyen d'organiser un ensemble de processus pour résoudre des problèmes complexes. En effet, traiter un problème de robotique ne se résume généralement pas à l'utilisation d'un planificateur. Un grand nombre de processus sont indispensables à la réalisation de tels problèmes, et qui peuvent être regroupés en six catégories :

- **Planifier** : c'est-à-dire créer des plans d'actions.

- **Agir** : exécuter un plan. Cela inclut la transcription des actions du plan en ordres moteurs par exemple.
- **Observer** : construire un modèle de l'environnement cohérent et utilisable à partir des données des capteurs.
- **Contrôler** : vérifier que l'exécution d'un plan correspond bien à ce qui était prédit, et déclenche une réparation le cas échéant.
- **Délibérer** : vérifier l'état des objectifs et voir s'ils sont toujours pertinents. Choisi également quel objectif devrait être traité, et vérifie que les plans calculés sont cohérents avec les objectifs choisis.
- **Apprendre** : mettre en place les moyens permettant de s'adapter et d'améliorer les performances tout au long de l'expérience.

2.2 Systèmes de décision déterministe

Les systèmes de décision déterministe sont utilisés pour résoudre des problèmes avec de multiples objectifs et sont appliqués dans de nombreux domaines comme : la manufacture [Hung and Leachman, 1996], l'exploration [Kuipers and Byun, 1991, Ceballos et al., 2011], la robotique domestique [Galindo et al., 2008] ou les véhicules autonomes, terrestres, aériens et spatiaux [Jónsson et al., 2000, Soto et al., 2007, McGann et al., 2008].

Les travaux présentés ne s'inscrivent pas dans le paradigme probabiliste que nous avons choisit. Par conséquent, la plupart de ces systèmes produisent des plans, peu fiables et qui nécessitant d'être réparés. Cela a pour conséquence que leur capacité prédictive est faible.

Cependant, certains de ces travaux proposent des solutions innovantes en particulier sur la structuration des objectifs, et sont capables de traiter de multiples ressources. Quelques travaux traitent en outre des problèmes proches du nôtre, avec un grand nombre de caractéristiques communes, et donnent des pistes intéressantes, proches des choix que nous avons fait dans ce travail.

2.2.1 Traiter de multiples objectifs, possiblement contradictoires

De nombreux travaux sont menés afin de permettre à un système décisionnel de dégager un plan dont l'exécution permet de résoudre un objectif donné. Ces travaux explorent différents techniques de résolution (heuristiques [Bonet and Geffner, 2001], modèle hiérarchisé [Saaty, 2008] etc.), s'attaquent à de nombreuses contraintes ou intègrent des modèles de plus en plus complexes (logique temporelle [Khatib et al., 2001], intégration d'ontologies [Záková et al., 2011], multi-agents [Brenner and Nebel, 2009], etc.), et donnent des méthodes efficaces pour réaliser un objectif défini. Cependant, la plupart de ces systèmes de décision disposent d'une capacité limitée à résoudre des objectifs multiples. Ces systèmes sont en effet conçus pour résoudre l'ensemble des buts donnés, sans concessions possibles.

L'aspect contradictoire de deux buts peut alors mener à des situations où le planificateur ne retourne aucun plan [Hawes, 2011]. Ce problème est d'autant plus important que nous pensons qu'un problème complexe, tel que celui auquel nous nous intéressons,

introduit naturellement des situations où des objectifs sont contradictoires et où tous les objectifs ne peuvent être résolus. La notion clé de cet aspect du problème est la capacité d'un agent à être autonome dans le choix de ses objectifs, imposant donc de dégager une méthode permettant de choisir quels objectifs réaliser et dans quel ordre.

Les travaux de [Pollack and Horty, 1999] proposent un système décisionnel réagissant en cas de conflit entre des buts menant à l'impossibilité de les résoudre tous. Le système proposé détecte les conflits et les résout en abandonnant un des buts en conflit, qui ne sera pas résolu au profit des autres.

[Scheutz and Schermerhorn, 2009] présente un système de sélection des objectifs basé sur leur réussite passée. Basé sur l'observation faites dans des travaux de psychologie [Clore et al., 2001], que les humains peuvent utiliser les affects pour gérer de multiples contraintes, ce travail propose d'utiliser ce concept pour un robot interagissant socialement avec des humains. Les auteurs proposent ainsi de garder en mémoire les réussites et les échecs des tâches entreprises afin de déterminer leur priorité. Plus une tâche fonctionne souvent (et donc produit un stimuli positif, plus cette tâche sera utilisée. Ce choix simple et rapide permet dans un contexte d'interaction et de coopération avec des humains d'améliorer l'efficacité de l'agent.

Notre travail se plaçant dans le domaine probabiliste, le système de récompense établi par les systèmes de décision markoviens nous permet de gérer facilement les priorités entre un ensemble de solutions. À chaque objectif est associé une quantité de récompense et les algorithmes de décision maximisent l'espérance de la récompense obtenue.

2.2.2 Introduction de multiples ressources

Certain systèmes de délibération prennent en considération des ressources comme le temps [Georgeff and Lansky, 1987, Ghallab and Laruelle, 1994, Lemai and Ingrand, 2004], permettant de modéliser des buts dont la réalisation dépend de la notion de temporalité, ou de l'énergie [Rabideau et al., 1999]. Ces systèmes peuvent alors résoudre de nouveaux problèmes, liés à la problématique de l'autonomie, grâce à la modélisation de nouveaux objectifs.

L'introduction des ressources peut également être à la base d'un processus de contrainte. Avec un scénario proposant pour un robot de maximiser le nombre de tâches exécutées jusqu'à l'épuisement de sa batterie ou pour une durée donnée. Chaque action ayant un coût propre, le planificateur cherche à connaître le coût demandé par chaque tâche et détermine le meilleur moyen de rentabiliser les ressources dont il dispose. Le travail présenté par [Hanheide et al., 2010] introduit un système de sélection des objectifs dont le principal facteur de choix est le temps pour réaliser la tâche demandée.

Le travail de [Haslum and Geffner, 2014] introduit la gestion du temps en plus de multiples ressources, dont le niveau décroît de façon monotone. À chaque action est associée une durée et un coût pour les différentes ressources. Le planificateur proposé trouve une solution optimale au problème posé, et en choisissant le plan requérant le moins de temps possible. Les autres ressources sont utilisées afin de définir des contraintes sur la réalisation des actions.

Le problème posé par [Wawerla and Vaughan, 2007] introduit la batterie comme ressource rechargeable pendant l'expérimentation. Un robot mobile récolte des ressources (du minerai par exemple) et a une batterie de capacité limitée, qu'il est possible de

recharger à un poste fixe. Ce travail explore différents comportements permettant de déterminer le moment opportun pour aller se recharger.

[Coddington and Luck, 2004] propose une vision des ressources directement associées aux objectifs, à l'aide de *drives*. Ces travaux seront étudiés plus tard, Section 2.2.4.

Pour notre part, nous avons fait le choix d'inclure des ressources dont l'évolution n'est pas monotone (mis à part le temps, de façon évidente). Cette propriété, bien que contraignante, nous permet de considérer des ressources pouvant être renouvelées (recharger la batterie du robot par exemple). Cela nous permet à la fois d'utiliser des ressources comme contraintes (des dates butoirs essentiellement) dans un scénario sans limite de temps à priori, ainsi que la problématique du niveau de la batterie qui ne doit pas se vider.

2.2.3 Objectifs structurés

La plupart des systèmes de décision s'attaquent à un problème où tous les objectifs posés peuvent être atteint depuis la situation initiale. Cependant, dès lors qu'un objectif est complexe et long à réaliser, il est intéressant de le découper en un ensemble de sous-buts structurés. Un objectif peut être découpé en étape à réaliser, et/ou être sous la forme d'un arbre à embranchement. Réaliser plusieurs objectifs ainsi modélisés implique alors de prédire, lors de la réalisation des sous-buts courants, les effets de leurs résolutions sur l'état futur du problème, mais aussi en termes d'apparition de nouveaux sous-buts.

Pour prévoir l'exécution d'une future tâche (et plus encore lorsque des objectifs peuvent être contradictoires), la plupart des travaux considèrent l'ensemble des buts comme un unique objectif, composé de sous-tâches. Cet objectif est structuré de manière à exprimer les différents enchaînements possibles de tâches.

L'introduction de la logique temporelle [Emerson, 1990] répond à ce besoin en proposant un langage permettant d'exprimer des enchaînements logiques de tâches, des conditions à remplir avant d'effectuer une tâche, ou des états à éviter. Dès lors, résoudre des problèmes écrits avec ce langage amène à considérer des objectifs complexes, composés d'une multitude de sous-tâches à enchaîner, et de les résoudre. C'est le cas dans les travaux de [Kress-Gazit et al., 2009], qui produit un automate donnant un comportement à adopter pour répondre à un objectif représenté à l'aide de la logique temporelle.

Pour répondre à ce besoin de structure, nous avons choisi des objectifs modélisés à l'aide de machines à nombre d'état fini, nommés *motivations*. Ces motivations modélisent chacun un "grand" objectif en définissant des états motivationnels (les états de l'automate) à partir desquels des tâches simples sont réalisables (les transitions de l'automate). Les enchaînements de tâches à réaliser et d'états permettent ainsi de modéliser un objectif complexe. Le Chapitre 3 est dédié à la présentation de ce modèle d'objectif.

2.2.4 Objectifs bouclants

Nous souhaitons traiter des problèmes de "longueur non finie" et des tâches répétitives. Un tel choix est nécessaire à la réalisation d'un scénario où un agent est en autonomie complète : un ensemble de tâches à accomplir un nombre indéterminé de fois et aucune limite de temps explicite.

Ce type de scénario est présent dans la littérature, et est généralement inspiré des expéditions sur mars à l'aide de rover. Ces scénarios mettent en place un robot dans un environnement où il doit effectuer un certain nombre de tâches. Il s'agit en général d'aller chercher des minerais, de faire des analyses ou de communiquer des résultats. Ces tâches reviennent chaque fois qu'elles sont exécutées et ne sont pas ordonnées (mais peuvent être priorisées). Le système décisionnel doit alors définir l'ordre dans lequel satisfaire les tâches. De nombreux travaux traitent ce problème [].

Comme dans le cas des objectifs complexes et enchaînés, le problème de la structuration des objectifs apparaît. Une solution au problème ne peut pas être envisagée comme un plan à exécuter, mais doit intégrer au travers d'une architecture un moyen d'exprimer des objectifs récurrents. Cette architecture est alors chargée de résoudre le problème courant en prenant en compte l'état futurs des objectifs. De plus, la production de plans devient continue, puisque de nouveaux objectifs sont ajoutés au fur et à mesure de l'avancement du problème.

Les travaux de [Knight et al., 2001], introduisent le concept de planification continue. Pour répondre rapidement à un changement d'état non prévu ou à une modification des objectifs pendant l'exécution d'un plan, le système *CASPER* vérifie à chaque pas si le plan est toujours valide (il le répare dans le cas contraire), modifie le plan actuel si les besoins ont changés et rallonge l'horizon de planification. Cependant, si ce système permet d'éviter de longues périodes de calcul entre chaque planification classique, il ne montre pas de capacité à anticiper l'apparition des prochains objectifs.

Nous nous intéressons à présent à un travail qui a retenu notre attention par la similarité des problématiques posées et par les solutions proposées. Dans la présentation de leurs travaux, [Coddington and Luck, 2003] s'attaquent à un problème d'un *mars rover* totalement autonome, c'est-à-dire en termes de décision (aucune requête posée de façon extérieure) et en termes de ressources (le rover a une batterie dont le niveau baisse et doit empêcher qu'elle se vide). Tout comme dans notre problème, l'expérience n'a pas de fin, et le robot doit effectuer continuellement un nombre de tâches scientifiques. Le robot présenté doit se déplacer sur la surface de Mars afin d'accomplir l'ensemble de ses missions : conserver quoi qu'il arrive un niveau d'énergie minimal, prendre des images, les transmettre à un appareil renvoyant les données sur terre et pour finir appliquer un certain nombre de mesure de sécurité.

Dans leur travail, les objectifs sont appelés *motivations* ou *drives*. Puisque nous avons choisi d'appeler également la structure de nos objectifs *motivations* et pour éviter les confusions, nous parlerons ici de *drives* uniquement.

Un *drive* est une structure de donnée simple, regroupant un objectif (un état à atteindre) et une "ressource" correspondant à l'objectif. Un *drive* détermine à partir de quelle valeur de la ressource associée l'objectif devient actif. Pour le *drive* de la conservation de la batterie par exemple (associée au niveau de la batterie), lorsque le niveau d'énergie passe sous un seuil donné, le but consistant à recharger la batterie s'active. Lorsque le niveau de la batterie passe à nouveau au dessus du seuil, le but se désactive. Pour les autres objectifs, respectivement prendre des images et les transmettre, les ressources associées sont le temps (l'objectif devient activé après une certaine période sans prendre d'image) et l'espace de stockage des images (quand celui-ci devient presque saturé, le robot doit envoyer les images afin de vider son espace de stockage). Puisque l'impact sur les ressources de chaque action du robot est connu, le système peut prévoir si un plan actuel déclenchera un des *drives* et donc commencer en amont à planifier les futurs but qui s'activeront [Coddington, 2007].

Les pistes pour la modélisations des objectifs est intéressante, et en particulier le fait de lier l'activation ou la désactivation des buts à un niveau de ressource. Cependant, et en regard à ce que nous avons étudié dans cette partie, nous considérons qu'un objectif ne peut pas se résumer à l'accomplissement d'une simple tâche. Nous pensons qu'il est trop limité de n'activer les buts que par des niveaux de ressources et nous souhaitons pouvoir activer un but par enchaînement, par exemple. Pour ce qui est de la partie planification, si l'activation de futurs buts est prédite et prise en compte, le planificateur organise les tâches afin de produire le plan le plus court, sans prendre en compte la priorité ou l'urgence des objectifs. Ils évoquent par exemple qu'il peut arriver que la batterie du *rover* se vide à cause d'un mauvais agencement de plan. Nous souhaitons pour notre part être en mesure de prendre en compte tous les effets des solutions résolvant les tâches afin d'éviter des plans qui pourraient être incohérents.

2.2.5 Conclusion

Pour conclure, les systèmes de décision déterministes apportent un ensemble d'outils intéressants, en particulier pour la gestion du temps et des ressources et des pistes intéressantes pour la modélisation d'objectifs structurés en lien avec les ressources. Remarquons enfin que les méthodes pour structurer, intégrer et gérer des objectifs reste un champ de recherche ouvert, et de nombreux travaux continuent de se pencher sur ces questions [Ingrand and Ghallab, 2014].

Cependant, les systèmes présentés utilisent des méthodes déterministes et donc imprécises. Si les méthodes de réparation de plan permettent à ces systèmes d'être utilisés, ces probables réparations rendent impossible le calcul de prédictions fiables sur les effets des plans. Au vu du problème que nous souhaitons traiter, ce problème ne semble pas pouvoir être surmonté. Pour finir, un grand nombre de travaux présentés ne sont que difficilement convertibles vers le paradigme probabiliste que nous avons choisi, à cause de la différence parfois radicale des modèles utilisés.

2.3 Approche probabiliste

Les systèmes de décision probabiliste sont également utilisés dans les domaines de la robotique [Ingrand and Ghallab, 2014, Renaudo et al., 2015] et de l'industrie [Pettersson, 2005], la robotique domestique [Schiffer et al., 2012], mais également dans le domaine de l'économie [Chakrabarti, 1999, Sloan, 2007].

L'environnement dans lequel un robot évolue est trop complexe pour pouvoir construire un modèle exact le représentant. Pour remédier à ce problème, une technique consiste à intégrer de l'incertitude dans la représentation en utilisant un modèle stochastique. Pour cela, il est possible de représenter le problème de façon approximative, qui omet quantité de détails, et de compenser ce manque de précision par l'incertitude [Bresina et al., 2002].

Les représentations probabilistes sont des représentations stochastiques dans lesquelles l'incertitude des actions est modélisée à l'aide d'une fonction de transition, donnant la probabilité de passer d'un état à un autre en exécutant une action. Pour une majorité des travaux utilisant ce type de modélisation, il s'agit de *processus de décision Markoviens* (MDP). Son aspect plus réaliste lui permet de calculer des plans complets (c'est-à-dire donnant l'action à effectuer pour chaque état possible), appelés *politiques*. Ces dernières sont robustes et ne nécessitent jamais d'être réparées. En plus de leur robustesse, il est possible de prédire précisément le comportement des politiques calculées [Kolobov, 2012].

En revanche, les processus de décision probabiliste sont très sensibles à la taille du problème à traiter. Le fait d'effectuer une recherche complète (sur l'ensemble des états) oblige à énumérer tous les états possibles et à placer en mémoire de grandes quantités de données, ce qui peut faire rapidement exploser le temps nécessaire pour calculer une politique, ou, de façon plus grave, saturer la mémoire (voir Section 3.2.1). Afin de pallier ce problème, de nombreuses méthodes, hiérarchiques ou factorisées, ont été développées. De plus, certain travaux n'utilisent pas de méthodes complètes et effectuent des recherches locales, proches des méthodes de décision déterministes, en conservant le paradigme probabiliste.

Pour finir, nous notons que le langage de description de problème RDDL (Relational Dynamic Influence Diagram Language) [Sanner, 2011] semble pouvoir décrire le type de problème auquel nous nous intéressons. Cependant, à notre connaissance, aucun planificateur ne permet de traiter les problèmes exprimés dans ce langage en totalité. Ces planificateurs omettent chacun une partie de ce que peut exprimer ce langage.

2.3.1 Multiples objectifs

Intégrer une multitude d'objectifs n'est pas un défi pour les systèmes de décision markoviens. Le système de récompense pour les états ou actions déclarées utiles permet de définir un ensemble d'objectifs. La fonction de récompense résultante est traitée en intégralité et le MDP retourne une politique qui maximise l'espérance mathématique des gains de récompense.

Cependant, certain travaux considèrent d'autres facteurs d'optimisation que la maximisation de l'espérance des gains de récompense. Il s'agit généralement de définir plusieurs objectifs séparément, et de chercher à équilibrer leur progression. Ainsi, afin de maîtriser la réalisation de plusieurs objectifs, les travaux de [Chatterjee et al., 2006] proposent de définir non pas une, mais un ensemble de fonctions de récompense, définissant chacune un objectif. Les objectifs ainsi séparés, ils proposent une méthode permettant d'obtenir une stratégie, constituée à l'aide d'un historique des états précédemment parcourus, permettant de satisfaire de façon égale l'ensemble des objectifs. La stratégie retournée est une distribution de probabilité sur les actions qu'il est possible de réaliser. En produisant une politique mixte évolutive, cette méthode permet d'atteindre un *optimum de Pareto* [Debreu, 1954] pour les objectifs considérés. Nous rappelons qu'une stratégie est Pareto-optimale lorsque aucune autre stratégie ne possède, pour chaque objectif, une meilleure espérance des gains de récompense.

2.3.2 Intégration de ressources

La modélisation des ressources dans les MDPs est un défi car celles-ci se combinent assez mal avec la notion d'état que ce formalisme propose, les états étant uniques et dénombrables, alors que les niveaux de ressources sont généralement donnés par une valeur numérique. Cependant, plusieurs pistes ont été explorées pour pallier cette difficulté. La première consiste à discrétiser les ressources pour avoir à nouveau un ensemble d'états unique, de la même manière que dans les *processus de décision markoviens partiellement observables* (POMDP) pour traiter l'incertitude sur les états. La deuxième piste consiste à conserver les valeurs numériques des ressources et à abandonner les recherches complètes au profit d'une méthode de recherche locale. Dans ce cas, les ressources ne sont généralement pas renouvelables et le temps est utilisé uniquement comme une contrainte.

Les ressources renouvelables et la ressource de temps sont toutes les deux abordées dans la littérature, mais apportent des problématiques différentes. Les ressources sont généralement multiples et bornées, tandis que le temps est unique et non borné (il est n'est borné uniquement lorsqu'il joue le rôle de contrainte). De ce fait, des travaux traitent spécifiquement un seul des deux problèmes et les solutions proposées sont rarement compatibles entre elles. Nous nous intéressons ici en priorité aux travaux traitant les deux à la fois. Cependant, nous pouvons tout de même noter quelques travaux ne traitant que le temps ou uniquement des ressources autres que le temps. Le travail de [Mausam and Weld, 2008] passe par exemple en revue les MDPs *hybrides*, c'est-à-dire combinant un espace d'état normal, discret et dénombrable, et un espace infini pour les ressources (ici, uniquement le temps). Ce travail s'intéresse au temps sous toutes ses formes, actions avec durée variable et stochastique, en plus de la gestion d'actions simultanées. Les travaux de [Guestrin et al., 2006] portent, eux, sur la gestion de multiples ressources, sans considération de la notion de temporalité. Il propose également d'utiliser des MDP hybrides, mais également de les factoriser afin d'augmenter les performances des algorithmes de décision.

Les travaux de [Le Gloannec et al., 2008] proposent une solution à un problème d'optimisation pour l'utilisation de multiples ressources. L'algorithme proposé permet à l'agent de savoir le moment opportun pour arrêter la tâche qu'il est en train de réaliser, si les ressources lui manquent; il sacrifie alors la récompense apportée par la tâche. Il permet également d'adapter sa consommation en ressources pour réaliser les tâches qui lui sont confiées, par exemple en optant pour une réalisation plus longue au profit d'une consommation en batterie inférieure. À chaque étape de chaque tâche, le robot choisit entre plusieurs profils pour faire progresser la tâche, chaque profil ayant une influence sur la récompense rapportée par la tâche lorsqu'elle se termine. Cependant, les travaux présentent des ressources qui ne peuvent qu'être consommées, et non renouvelées.

L'intégration de ressources dans les processus markoviens peut aussi prendre la forme d'un coût à minimiser

C'est le cas dans les travaux de [Svorenova et al., 2013], qui propose la résolution d'une tâche répétitive, définie à l'aide de la *logique temporelle linéaire*, un aller-retour entre deux emplacements défini à l'aide de la logique temporelle. En utilisant une stratégie avec mémoire, il devient possible d'obtenir un comportement exécutant de façon optimale la tâche donnée en minimisant le coût en ressource accumulé de la tâche.

Tout comme pour les systèmes déterministes, des travaux probabilistes ce sont intéressés à la résolution de problèmes locaux, limités dans le temps et contraints par des ressources. Ces travaux utilisent des méthodes de recherche locale plutôt que les méthodes globales classiques. C'est le cas dans [Meuleau et al., 2009], qui traite un problème d'autonomie proche du nôtre. L'objectif de ce travail est de permettre à un Mars rover de résoudre un ensemble de tâches pendant les périodes où la communication avec la Terre est impossible. Lors de ces périodes, le robot dispose de ressources limitées pour agir, qui sont le temps et l'énergie disponible du *rover*. Ils souhaitent pour cela générer un plan conditionnel permettant de maximiser la récompense obtenue en exécutant des tâches scientifiques jusqu'à épuisement des ressources. Pour guider l'exploration du rover et la recherche, ils utilisent les contraintes en ressources pour déterminer quelle partie de l'espace de recherche devrait être explorée. Cependant, les ressources ne peuvent pas être rechargées, ce qui permet uniquement un travail d'optimisation pour l'utilisation des ressources disponibles. Le modèle présenté par [Bresina et al., 2002] traite un problème

très similaire, dont le modèle du monde est déterministe, mais dont les coûts en ressources sont stochastiques. Ils utilisent également une planification locale, à la différence qu'ils développent une méthode pour calculer un plan conditionnel. Ce plan contient un ensemble de plans de secours lui permettant d'agir dans le cas où le rover consomme plus d'énergie qu'espéré par exemple.

2.3.3 Solutions hiérarchisées et factorisées

Les travaux présentés précédemment et qui utilisent des solution complètes (calculant donc des politiques) font toutes face au même problème : l'explosion de l'espace d'état, et donc du temps et de l'espace nécessaire pour résoudre le problème, lorsque la taille du problème augmente. Si les solutions locales utilisent un ensemble d'heuristiques pour limiter l'exploration des algorithmes, les méthodes complètes deviennent assez rapidement inutilisables, et inapplicables pour des problèmes réalistes.

Pour combattre ces limitations, des méthodes qui introduisent de la structure dans le problème ont été proposés. Nous nous intéressons ici à deux ensembles de techniques :

- Les techniques qui, en cassant l'objectif en une multitude de sous-buts, et calculent autant de politiques (définies généralement sur des espaces plus réduits). Les politiques calculées sont ensuite utilisées comme *macro-actions* pour résoudre un problème plus vaste. Cette méthode a son équivalent dans le paradigme déterministe avec par exemple les Réseaux de Tâches Hiérarchiques (HTN) [Nau et al., 2003].

- Les méthodes pour factoriser l'ensemble du modèle du problème. Ces techniques abandonnent les structures de données des MDP classiques (à savoir des ensembles d'états, d'actions, fonctions de transition, etc.) pour de nouvelles structures. Celles-ci permettent de supprimer les redondances des représentations classiques, et donc de condenser le modèle des problèmes traités.

Notons que peu de systèmes à notre connaissance utilisent à la fois des ressources et des représentation hiérarchisées ou factorisées.

Les *options* sont un moyen d'introduire des routines dans la résolution d'un problème [Sutton et al., 1999]. L'objectif ici est d'optimiser les temps de calcul en donnant une solution prédéfinie pour un sous-ensemble d'état. Pour un MDP les utilisant, les *options* sont des politiques pouvant être exécutées à partir d'un nombre limité d'états. Chaque *option* étant définie sur un domaine limité, il est possible de calculer, pour chaque état limitrophe de ce domaine, la probabilité d'y arriver en utilisant cette *option*, ainsi que la récompense obtenue en moyenne pour y arriver, en fonction d'un état de départ. Pour finir, une *option* obtient le même modèle qu'une action et peut être utilisée en lieu et place des actions par les algorithmes de recherche.

La méthodes des tâches hiérarchiques (*task hierarchy* en anglais) améliore les *options* en proposant de découper hiérarchiquement l'objectif du MDP en sous-objectifs [Dietterich, 2000a], résolus séparément et dont la solution s'utilise comme une *option*. Les sous-objectifs sont résolus en créant des *sous-tâches* (qui ressemblent aux *options*), définies sur un domaine d'états réduit, un ensemble d'états buts et une sous fonction de récompense. Les sous-tâches sont calculées à l'aide de techniques classiques de décision (contrairement aux *options* qui sont définies à la main) et en utilisant les *sous-tâches* de niveau inférieur précédemment calculées. De la même manière que pour les *options*, l'exécution des *sous-tâches* est modélisée sous la même forme que des actions. De plus,

les méthodes de résolution utilisées prennent en compte que les *sous-tâches* ne s'exécutent pas en un seul pas de temps, mais sur plusieurs pas de temps.

Ces deux solutions sont intéressantes car elles permettent d'utiliser des opérateurs de haut-niveau, achevant chacun un sous-problème, pour résoudre un problème plus grand. Elles introduisent de la structure dans le problème, permettant ainsi de réduire le nombre d'actions et donc de réduire le temps et l'espace nécessaire pour résoudre le problème posé. Cependant, le calcul de sous-solutions et leur modélisation sous la forme de *macro-actions* peut être coûteux. De plus, par l'introduction de la décomposition du problème, l'optimalité de la solution n'est plus garantie. La notion d'*optimalité hiérarchique* (hierarchical optimality en anglais) [Kolobov, 2012] est alors utilisée pour parler des meilleures solutions possibles compte tenu de la décomposition.

Discutons à présent des techniques permettant d'exploiter les redondances présentes dans les représentations des problèmes pour optimiser leur résolution.

Une représentation entièrement explicite est nommée *extensional representation*, elle consiste à lister, comme des les MDPs classiques, la liste de tous les états possibles. Une *intentional representation* décrit ces mêmes états en décrivant ses propriétés. Lors de la construction de la table de transition d'un problème utilisant une *extensional representation*, les actions n'étant pas conditionnées et n'agissant que sur un ensemble réduit de propriétés des états vont créer des redondances. Deux transitions pourront être pratiquement identiques, avec seulement le changement d'une propriété de l'état de départ n'ayant aucune influence dans la résolution de l'action.

Les MDPs factorisés [Boutilier et al., 1999] utilisent une *intentional representation* pour décrire le modèle du problème. Ils permettent de faire disparaître les redondances en utilisant des descriptions factorisées pour l'espace d'état, les actions, la fonction de transition ainsi que la fonction de récompense. Une solution pour représenter le problème de manière factorisée est d'utiliser des Opérateurs Probabilistes STRIPS (PSO) [Boutilier et al., 1999], une représentation inspirée du langage STRIPS, servant de base à la description des problèmes déterministes. De nouvelles techniques de décision peuvent ensuite être utilisées afin de résoudre les problèmes sous leur forme factorisée.

2.3.4 Conclusion

En conclusion, les travaux sur les systèmes probabilistes répondent à l'ensemble des problématiques auxquelles nous nous intéressons, mais de manière séparée. L'intégration de ressources semble difficile, entraînant une lourde augmentation de la taille du problème. De plus, si la factorisation des problèmes est possible, il est difficile de voir comment elle peut s'effectuer avec la présence de ressources, puisqu'elle n'introduit à priori pas de redondance. Cependant, des travaux considèrent des ressources non renouvelables, d'autres les considèrent comme des facteurs d'optimisation, et d'autres encore utilisent des méthodes de recherche locales. Le choix de la manière de représenter les ressources a donc un impact fort sur la direction que prennent les différents travaux.

D'autre part, les méthodes permettant de hiérarchiser les objectifs d'un problème nous semble prometteuses. Cette solution permet de réduire l'espace sur lequel effectuer des calculs, tout en permettant de structurer les objectifs. Nous explorerons en détail cette problématique dans le Chapitre 3 pour la conception des objectifs et Chapitre 4 pour leur résolution sur des espaces factorisés.

2.4 Architectures délibératives

Les architectures délibératives sont des systèmes dont l'objectif est d'intégrer de nombreuses fonctionnalités, qu'un planificateur seul ne pourrait faire cohabiter, afin de résoudre des problèmes complexes. Ces systèmes sont utilisées particulièrement dans le cas de la robotique autonome [Ingrand and Ghallab, 2014], grâce à leur faculté d'agencer et de hiérarchiser des ensembles de processus indispensables en robotique, comme par exemple assurer la conversion entre les données physiques des capteurs et un modèle du problème plus symbolique, utilisable par le planificateur. De plus, conscients des difficultés rencontrées lors du passage à l'expérimentation réelle, la gestion de l'incertitude et des erreurs qui en découlent (modèles imparfaits, échec de la planification, etc.) est traitée.

Cependant, au-delà des problématiques auxquelles elles s'attaquent, il est difficile de faire des généralités sur les architectures délibératives. Chacune d'entre elles aborde les problèmes posés de façon différente et le modèle architectural proposé est donc chaque fois différent.

Pour étudier ces systèmes, nous allons nous focaliser sur les architectures apportant des réponses innovantes et traitant les caractéristiques de notre problème.

2.4.1 Architecture hiérarchique

Les architectures hiérarchiques sont une forme courante de système délibératif et reposent généralement sur trois couches, la première, la plus abstraite, gère les objectifs et conçoit des plans à partir d'un modèle évolué, la seconde est chargée de traduire les plans créés pour pouvoir les exécuter et la troisième, la plus primitive, agit directement avec les capteurs et les moteurs [Gat et al., 1998, Alami et al., 1998]. Il existe de nombreuses variations de ce modèle, concernant le rôle exact de chaque couche. Cependant, la couche la plus élevée gère toujours les objectifs et planifie leur résolution. Elle manipule un modèle ayant un haut niveau d'abstraction et est généralement la seule ayant accès à la représentation du temps [Ingrand et al., 2007]. La couche intermédiaire est chargée de construire le modèle utilisé par la couche supérieure en utilisant les données des capteurs [Carbone et al., 2008]. Il peut s'agir de traitement d'images ou de la construction d'une carte par exemple [Doherty et al., 2009]. Cette couche est également en charge de traduire les plans conçus pour qu'ils soient exécutés par la couche inférieure. Elle peut enfin être en charge de la supervision de l'exécution des plans, et éventuellement le réparer au besoin [Ingrand et al., 2007]. Pour finir, la couche inférieure exécute les ordres donnés par la couche intermédiaire, et renvoie aux couches supérieures les informations sur l'état d'exécution des actions ainsi que des données sur l'état du monde, récoltées par les capteurs. Cette couche a également un ensemble de fonction réactive, c'est-à-dire qui s'exécute sans délibération, permettant d'éviter des obstacles par exemple.

Dans notre travail, nous avons choisi de hiérarchiser le modèle du problème de manière originale, en utilisant deux modèles, intentionnel et opérationnel. Le premier modèle décrit les objectifs, leur fonctionnement et les tâches à réaliser, tandis que le second ne modélise que l'aspect physique du problème, c'est-à-dire ce qui peut être perçu ou ce sur quoi le robot peut directement agir.

La résolution du problème est alors séparée en deux couches. Tout d'abord le calcul des solutions résolvant les tâches des objectifs, en n'utilisant que le modèle opérationnel.

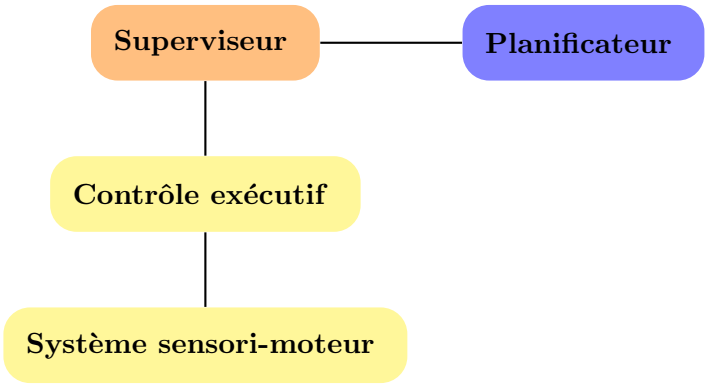


Figure 2.1: Schéma de l'architecture à trois couche et l'architecture que nous proposons. Présentation de l'architecture classique à trois couches.

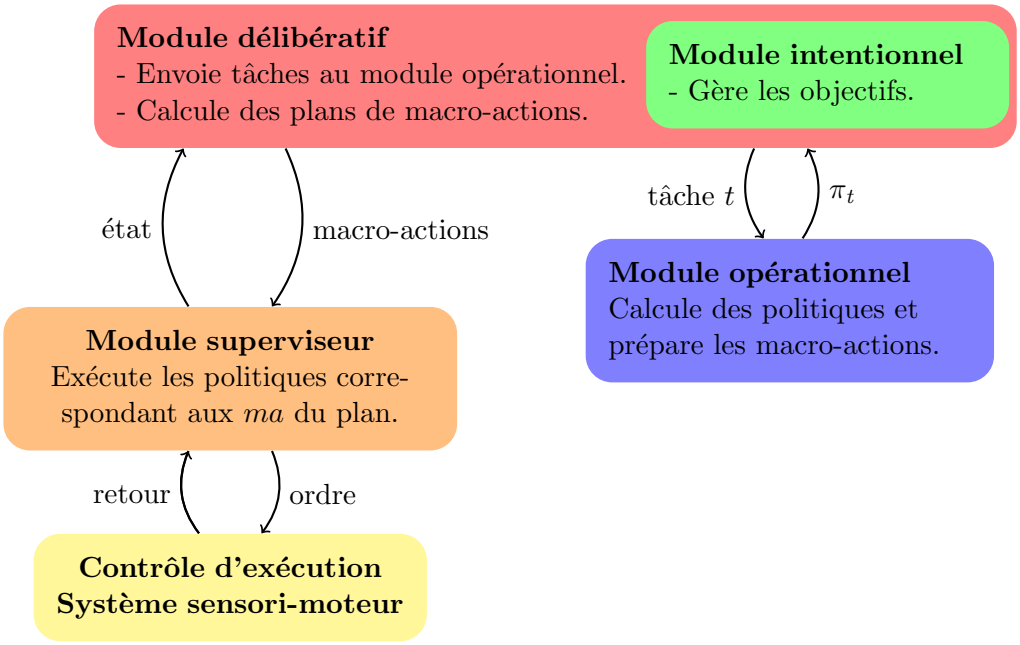


Figure 2.2: Présentation de l'architecture, composée des modules opérationnel, intentionnel, délibératif et du contrôle d'exécution / système sensori-moteur.

Ces solutions sont ensuite modélisées sous la forme de macro-actions, de la même manière que les HTN pour la décision déterministe, et ayant de bonnes capacités de prédiction. Et ensuite un processus de délibération effectue une recherche locale et détermine l'ordre dans lequel accomplir les tâches. Les Figures 2.1 et 2.2 illustrent respectivement les structures de l'architecture à trois couches et l'architecture que nous développerons dans ce mémoire de thèse. En comparant ces figures, nous pouvons voir que nous proposons de changer la planification en ajoutant un processus de délibération. La partie planification existe toujours avec le module opérationnel, mais il est commandé par le module délibératif.

C'est ce dernier qui, en planifiant à partir des calculs du module opérationnel, calculera une solution et l'enverra au superviseur.

2.4.2 Gestion des objectifs

La gestion des objectifs est, pour un agent autonome, l'ensemble des processus permettant de définir les objectifs actuels, les prioriser, et de vérifier que les plans actuels sont valides et cohérents avec les objectifs fixés [Vattam et al., 2013].

Certain travaux, comme ceux de [Muscettola et al., 1998], sont capables d'enchaîner la résolution de tâches. Une fois qu'un plan pour une tâche est calculé, le système est capable de déterminer à partir de la situation finale les prochains objectifs qu'il faudra résoudre, et de les planifier. Il évalue pour cela les ressources nécessaires pour l'accomplissement des futures tâches et estime si la fin du plan actuel remplit les conditions pour les tâches suivantes. Cependant, les plans résolvant les nouveaux objectifs sont ajoutés à la fin du plan actuel et le plan n'est revu en totalité que si le plan devient défaillant.

Les travaux de [Molineaux et al., 2010], proposent un système basé sur la faculté de choisir de façon autonome ses objectifs, déclenché par l'apparition d'événements inopinés, nommé *ARTUE* (Autonomous Response to Unexpected Events) et utilisé dans le domaine militaire. Le système utilise comme modèle une version de PDDL [Fox and Long, 2006] (un langage de description de problème issu de STRIPS) et un planificateur HTN. L'objectif de cette architecture est de détecter des facteurs cachés, dont il est possible d'observer les conséquences. Le travail cité donne l'exemple de l'observation d'un éclair, présageant qu'une tempête approche. Lorsque le système détecte un événement invalidant le plan en cours d'exécution, celui-ci cherche par abduction une explication à l'apparition de l'événement. Il produit une liste de suppositions expliquant les événements observés et en déduit une liste de prédicats qui sont ajoutés à l'état actuel. En fonction des nouvelles connaissances, le système peut générer des nouveaux objectifs suivant un ensemble règles pré-établies. Dans l'exemple de la tempête, le robot doit alors chercher un abri. La capacité de décision de l'architecture est limitée par le fait de ne pouvoir planifier qu'un seul objectif à la fois, en commençant par le plus prioritaire. Sa capacité de prédiction est alors fortement impactée. Ce principe d'autonomie a également inspiré [Wilson et al., 2014], pour la conception de robots sous-marins autonomes.

2.4.3 Conclusion

Les architectures délibératives proposent des solutions variées pour la résolution de problèmes de robotique autonome. La grande majorité d'entre elle traite au minimum la notion de temporalité et leur hiérarchie permet de s'attaquer à de vastes problèmes. De plus, leur fonctionnement en parfaite autonomie font d'elles de bons candidats pour traiter des problèmes sans limite de temps, grâce à la définition de nouveaux objectifs et grâce à la répétition des périodes de planification tout au long des expérimentations. Il s'agit là d'une technique intéressante que nous utilisons dans ce travail, permettant de formaliser l'utilisation de techniques de recherche locale pour la résolution de problèmes continus.

Cependant, la plupart de ces systèmes utilisent des méthodes de planification déterministes, limitant encore une fois drastiquement leur capacité de prédiction.

2.5 Conclusion de l'état de l'art

Dans cet état de l'art, nous avons étudié deux domaines liés : le domaine de la décision déterministe et les systèmes de décision probabiliste. Chacun de ces systèmes nous a permis d'explorer les solutions pour essayer de résoudre un problème avec des objectifs complexes structurés, avec autonomie dans le choix des objectifs à traiter, sans limite de temps explicite et avec la notion de ressource.

La littérature portant sur les systèmes déterministes nous a proposé des pistes sur la modélisation des objectifs, à l'aide de structures logiques simples, et des pistes pour le traitement de multiples objectifs. Les travaux traitant la problématique des ressources sont nombreux et développent des solutions variées, en termes de modélisation et de solution aux problèmes posés. Enfin, quelques travaux se sont attaqués à plusieurs problématique que nous avons soulevées, en introduisant de multiples buts structurés et avec la présence de ressources pour des scénarios d'autonomie sans limite de temps. Cependant, le paradigme déterministe n'est à notre avis pas suffisant pour résoudre le problème que nous avons posé. En effet, le modèle déterministe est trop rigide, et le modèle choisi sera donc toujours inexact, rendant inopérants les plans produits pour résoudre le problème. Ces plans doivent être réparés régulièrement, faussant ainsi la capacité de prédiction du système. Or, nous pensons que dans un contexte d'autonomie et avec de multiples objectifs à satisfaire, la capacité de prédiction est fondamentale.

Les systèmes probabilistes proposent au contraire des solutions complètes et donc il est possible de prévoir l'issue. Cependant, aucun travail de la littérature ne traite le problème que nous avons posé à notre connaissance. Des travaux apportent des solutions à différentes parties du problème, sur l'intégration des ressources ou la conception d'objectifs structurés. Nous avons également rencontré des planificateurs probabilistes locaux (c'est-à-dire calculant des politiques partielles, amenant à atteindre un but depuis un état de départ) et qui intègrent les ressources de façon convaincante. Cependant, l'utilisation seule de ces techniques nous semble inadaptée, encore une fois à cause de la capacité de prédiction limitée des solutions qu'elles calculent.

Néanmoins, de nombreuses techniques existent pour résoudre des problèmes de grande taille et offrant des solutions complètes (sous la forme de politiques), et qui ont par conséquent de bonne capacités de prédiction. Ces techniques permettent de décomposer le problème posé et de le structurer afin d'exploiter les régularités du modèle du problème. Cependant, aucune de ces techniques n'a à notre connaissance implémenté de multiples ressources.

Pour finir, les architectures délibératives proposent quant à elles d'associer un ensemble de processus afin de traiter un problème de grande taille et de gérer la planification et la réactivité. Les aspects que nous avons traités sont les systèmes hiérarchisés, qui permettent d'abstraire un modèle à partir de ses données élémentaires, et la formalisation d'un cycle permanent de planification, permettant d'utiliser des méthodes de recherche locales pour résoudre des problèmes continus.

Dans ce travail, nous proposons de partir du constat fait sur les systèmes probabilistes, leurs capacités à gérer de multiples objectifs et les méthodes pour hiérarchiser le problème posé, pour développer une architecture délibérative capable de traiter un problème multi-objectifs, autonome, traitant des ressources et sans limite de temps explicite. Nous nous inspirons des solutions déterministes, en particulier sur la construction d'objectifs structurés, afin d'introduire un nouveau modèle pour décrire des objectifs : les *motivations*, modélisées à l'aide de machines à nombre d'états finis. Nous formerons enfin un cycle

délibération / exécution qui nous permettra d'apporter une solution continue au problème sans limite de temps explicite posé.

Chapitre 3

Structurer de multiples objectifs

Contents

Lors d'un processus de décision, le choix du modèle des objectifs pose plusieurs défis :

- **Exprimer fidèlement les objectifs** : la capacité à exprimer des objectifs est étroitement lié à la façon dont le modèle du monde est constitué. Un objectif est souvent défini comme une portion de l'état du monde, qui peut être soit à atteindre, soit récompensée lorsque atteinte. Cependant, certain objectifs demandent parfois, pour être exprimés, des notions non présentes dans le modèle du monde, comme la temporalité des événements (faire apparaître l'événement A **puis** l'événement B).
- **Rester utilisable lors d'un passage à l'échelle** : la façon dont les objectifs sont exprimés peut changer radicalement les méthodes de résolution disponibles. Ces méthodes ont à leur tour une forte influence sur la taille de l'espace de recherche qu'il faudra explorer pour trouver une solution, en fonction de la taille du problème.

Notre travail se concentrant sur la décision probabiliste, nous présenterons les méthodes classiques de représentation d'un objectif, les états buts et des solutions plus originales comme *task hierarchy*. Nous présenterons enfin une méthode de représentation des objectifs originale, les *motivations*, que nous utiliserons dans ce travail.

3.1 Présentation de l'exemple récurrent

Pour illustrer ce chapitre et les suivants, nous nous baserons sur un exemple simple, qui est une version réduite du problème que nous avons traité pour nos tests expérimentaux. Il s'agit d'un atelier dans lequel un robot doit effectuer un certain nombre d'actions tout en conservant son énergie. Cet exemple a été constitué pour sa représentativité des tâches que peut accomplir un robot, navigation et manipulation, ainsi que pour l'opportunité de choix entre plusieurs tâches et de passer de l'une à l'autre.

3.1.1 Présentation

L'environnement de l'atelier est illustré Figure 3.1, et est découpé en neuf cases nommées de A à H, plus la case S. Le robot est muni pour se localiser d'une centrale inertielle, dont la précision se détériore en fonction du temps écoulé. Sa précision est modélisé par deux états distincts : **précis** et **imprécis**. Dans l'état précis, tous les déplacements du robot ont un taux de succès de 100%.

Lorsque la centrale inertielle est **imprécise**, le robot se déplace d'une case à l'autre depuis les cases de A à H, dans les quatre directions et avec une chance de succès de 70%, une probabilité de 5% de se déplacer sur une autre case adjacente (S n'est adjacente à aucune case), et reste sur sa case avec une probabilité de 15% si la case de départ possède quatre cases adjacentes, plus 5% pour chaque case adjacente en moins. À partir de la case S, il ne peut se déplacer que vers l'ouest et l'est, avec 50% de chances, et reste sur place avec 50% de chances. Les cases B et D permettent d'aller sur la case S en exécutant un déplacement vers le nord, avec les mêmes chances de succès que pour sortir de S.

Deux boîtes différenciées par leur couleur, une jaune et une bleue, placées initialement en A, peuvent être tenues (une seule à la fois) par le robot pour être amenées en D, sur le tapis roulant. Le robot doit être en A pour récupérer une boîte. Il a 75% de chances de réussir, et peut la déposer avec 85% de chances, les échecs n'ayant aucun effet. Lorsqu'il dépose une boîte sur le tapis roulant, celle-ci est emportée immédiatement et sort de l'atelier. Après son dépôt, la boîte déposée revient automatiquement en A, simulant le

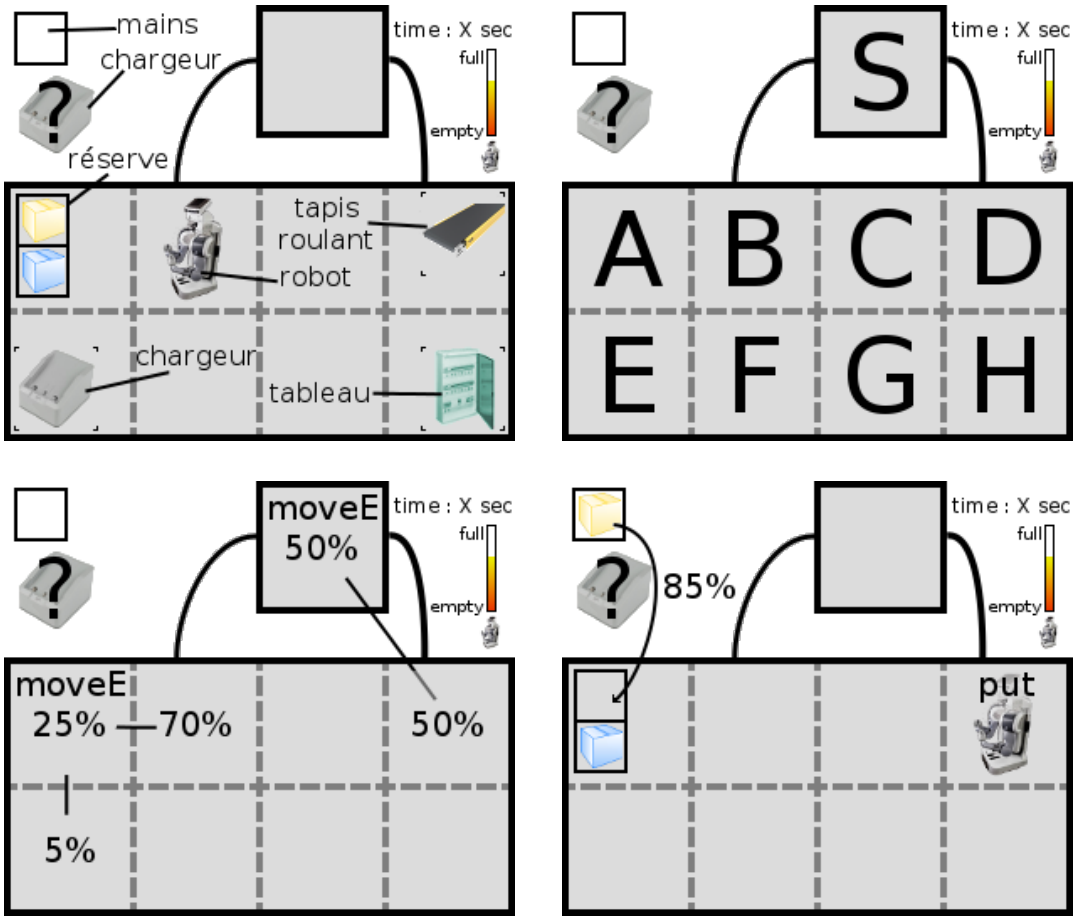
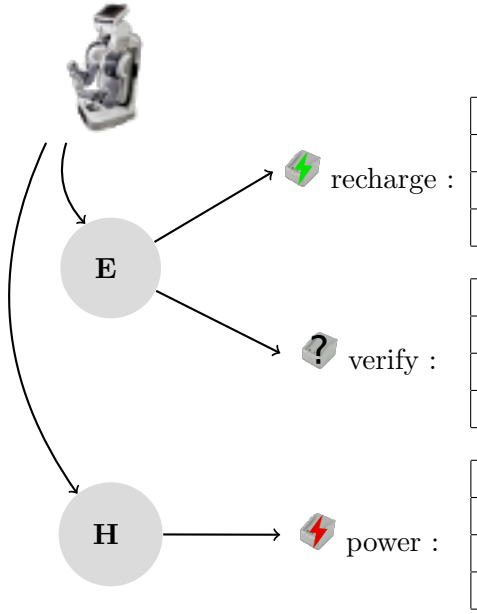


Figure 3.1: Représentation graphique de l'atelier réduit.

fait qu'il y a une grande quantité de chacune des boîtes, mais que le système n'en considère qu'une de chaque couleur à la fois.

Le robot peut enfin recharger sa batterie, en exécutant l'action correspondante depuis le chargeur sur la case E. Pour cela, le robot doit s'assurer de l'état du chargeur : celui-ci peut être dans les états *bad*, *good* et *unknown*. L'état *unknown* nécessite d'utiliser l'action *verify* sur la case E pour déterminer l'état actuel du chargeur. Dans l'état *bad*, le chargeur ne peut être utilisé et doit être réalimenté depuis le tableau de bord, sur la case H. Dans l'état *good*, le robot a 90% de chances de recharger ses batteries et le chargeur repasse à l'état *unknown*. Dans le cas contraire, le chargeur passe à *bas* et la batterie n'est pas rechargée. Dans l'état *bad*, le robot doit aller devant la tableau électrique sur la case H et doit utiliser l'action *power*, qui remet le chargeur dans l'état *good*. Ce fonctionnement est décrit dans la Figure 3.2. Le robot peut également se relocaliser depuis les cases C et G afin de rétablir la précision de la centrale inertielle. Cette action est réussie si, à son issue, le robot est toujours sur sa case de départ.

Le modèle de l'exemple inclut deux ressources : le niveau de la batterie, qui a une valeur comprise entre 0 et 100, et le temps, qui commence à 0 et qui représente le temps en unités de temps **ut** depuis le début de l'expérience.



ws \ *ws'*	Unknown	Bad	Good
Unknown	*n/a*		
Bad	*n/a*		
Good	0.90	0.10	0

ws \ *ws'*	Unknown	Bad	Good
Unknown	0	0.25	0.75
Bad	*n/a*		
Good	*n/a*		

ws \ *ws'*	Unknown	Bad	Good
Unknown	*n/a*		
Bad	0	0	1.00
Good	*n/a*		

Figure 3.2: Schéma et table de probabilité pour les actions *recharge*, *verify* et *power*.

3.1.2 Objectifs

Dans cet atelier, nous souhaitons que le robot dispose d'un comportement lui permettant de traiter plusieurs objectifs simultanément.

Maintenir le niveau de la batterie. La première fonction de notre de robot est de rester en état de fonctionner. Il doit pour cela recharger sa batterie régulièrement depuis la case E. Le robot perdra de la récompense lorsque sa batterie restera faible et en gagnera lorsqu'il la rechargera.

Cet objectif sera abrégé **Survie**.

Alterner sur le tapis roulant les boîtes *jaune* et *bleue*. Le travail du robot dans l'atelier consiste à emmener des boîtes depuis la case A jusqu'à la case D, où il faut les déposer sur le tapis roulant. L'objectif se découpe en deux parties : amener la boîte jaune, puis amener la boîte bleue, et recommencer. Le robot prend les boîtes une par une et doit effectuer cette tâche à l'infini. Il obtiendra de la récompense pour chaque boîte ramenée sur le tapis roulant (en respectant l'alternance jaune / bleu). En cas de non respect de l'alternance des boîtes, le robot ne reçoit pas de récompense.

Cet objectif sera abrégé **Boîtes**.

Éviter une zone restreinte. Pendant certaines périodes, la zone constituée des cases C et G est restreinte. Le robot devra éviter de passer dans cette zone à ces moments. Le robot subira une perte de récompense chaque fois qu'il se trouvera dans cette zone. La zone est restreinte 600 **ut**, puis libre d'accès pendant 3 000 **ut**, puis de nouveau restreinte, etc.

Cet objectif sera abrégé **Zone**.

Se relocaliser. La localisation du robot est déterminée par une centrale inertielle et sa précision décroit avec le temps. Le temps a donc une influence sur la variable **imuPrecision**, qui détermine l'efficacité des déplacements du robot. Pour naviguer convenablement, le robot devra se relocaliser régulièrement depuis les cases C ou G. Le robot pourra rétablir la précision de sa localisation 1200 **ut** après sa dernière localisation, lorsque la précision aura baissé, ou seulement après 720 **ut**, afin de prévenir de la baisse de précision.

3.2 État but

Dans le domaine de la décision probabiliste (MDP, POMDP), pour définir un objectif à atteindre et le modéliser, la technique la plus répandue est celle de l'état but (*goal state* en anglais). Celui-ci définit le couple (ws, r), un état du monde suivi d'une valeur de récompense (aussi appelée utilité). Des récompenses intermédiaires, plus faibles, peuvent également être fournies. Une fois l'objectif atteint, celui-ci disparaît et un nouvel objectif doit être fourni.

Limites

De notre point de vue, cette méthode n'est pas suffisante et ne permet pas de modéliser les objectifs dans leur ensemble, en particulier pour trois raisons.

Manque de précision des états buts. Lors de la conception d'un MDP, les modalités de la fonction de récompense peuvent varier. Courante et très simple, la fonction $R \rightarrow WS$ définit la récompense sur les seuls états. Lorsque l'agent exécute une action depuis ws, il obtient la récompense correspondante $R(ws)$. Si cette modalité est facile à mettre en place, d'autres permettent de profiter de l'expressivité offerte par les MDP. Nous pouvons citer $R \rightarrow WS \times A$ ou encore $R \rightarrow WS \times A \times WS$, qui associe la récompense obtenue aux transitions effectuées. Ainsi, une action a effectuée depuis l'état ws et dont l'effet change l'état courant pour ws' octroie la récompense $R(ws, a, ws')$.

De la même manière, utiliser des états buts ne permet pas d'utiliser l'expressivité maximum des objectifs permise par les processus Markoviens, permettant uniquement de cibler des états. Il est impossible, en outre, d'exprimer un objectif tel que : *utilise l'action a depuis l'état ws jusqu'à ce que l'état devienne ws'*. L'objectif étant ici autant un état que le moyen pour arriver à cet état. Utiliser comme objectif une transition complète $[ws, a, ws']$ permettrait de palier ce problème. Nous définissons ces transitions objectifs *transitions buts* $gt = [ws, a, ws']$.

Prenons par exemple un robot dont le but est de s'entraîner à lancer une balle depuis un point A jusqu'à un point B. Nous souhaitons alors définir un objectif consistant à réussir un lancer. Définir cet objectif comme un état, $(robot = A, balle = B)$, à atteindre ne convient pas. Avec seulement un état, impossible de savoir si le robot a lancé la balle. Si la balle a été déposée par le robot en B et qu'il se place ensuite en A, l'objectif serait alors considéré comme atteint (le robot pourrait ensuite entrer et sortir du point A pour accomplir l'objectif à de multiple reprises). En utilisant comme définition de l'objectif une *final transition*, $((robot = A, balle = A), lancer, (robot = A, balle = B))$, il est possible que préciser l'état précédant le lancer et l'action exécutée pour passer à l'état voulu, permettant cette fois de s'assurer que l'objectif a bien été réalisé.

Maintien. Prenons par exemple "Maintenir le niveau de la batterie", de notre exemple réduit. Modéliser cet objectif comme un goal state revient à dire qu'il suffit de remplir la batterie une fois et que l'objectif est atteint. Nous souhaitons au contraire que l'objectif ne disparaisse jamais, puisqu'il s'agit de maintenir la batterie à un bon niveau. Nous voulons alors que l'objectif soit actif lorsque la batterie est basse, et en attente lorsque celle-ci est haute. D'autre part, le fait que le niveau d'énergie passe de haut à bas est un événement crucial pour l'objectif. Nous souhaitons donc connaître la probabilité que la résolution d'un objectif quelconque, débuté alors que le niveau de la batterie est haut, fasse passer le niveau de la batterie à bas. Cela nous permettrait de prévoir à quel moment la batterie aura de nouveau besoin d'être alimentée.

Enchaînement. Une autre particularité que les états buts ont du mal à modéliser est l'enchaînement. Il n'est par exemple pas possible de modéliser l'objectif "Alterner sur le tapis roulant les boîtes *jaune* et *bleue*". En effet, pour modéliser la notion d'ordre entre les deux buts, il est nécessaire de différencier l'état d'avancement de l'objectif. Il est possible de contourner ce problème en ajoutant à la modélisation du problème une variable d'état représentant cet état d'avancement. Cette variable pourra s'appeler "objectifChainé" et prendra une des valeurs de l'ensemble $\{doitApporterJaune, doitApporterBleu\}$. L'état de cette variable passant de $doitApporterJaune$ à $doitApporterBleue$ lorsque la boîte *jaune* a été déposé sur la tapis, et inversement pour la boîte *bleue*. Cependant, l'inconvénient de cette technique est qu'elle multipliera la taille du problème par le nombre d'objectif à enchaîner et ceci pour chaque objectif chaîné. Cette augmentation de l'espace d'état rendant rapidement impossible la résolution de problèmes dû à l'explosion du temps de calcul. Nous souhaitons donc trouver une méthode permettant de décrire de tels objectifs en évitant la création de variables d'état supplémentaire.

3.2.1 Contournement naïf

Une solution naïve aux limitations présentées est d'aplatir le MDP, c'est-à-dire d'ajouter au modèle du monde autant de variables que nécessaire afin d'exprimer tous les objectifs simultanément.

Les objectifs de maintien peuvent être associés à une variable binaire pour les cas simples (la zone restreinte par exemple), ou à une variable à de multiples états pour les cas plus complexes (la localisation par exemple). Pour la zone restreinte, la variable passera à *restricted* lorsque la zone sera restreinte, et *freeAccess* dans le cas contraire. Dans le cas de la relocalisation, il est possible de définir une variable évoluant en fonction du temps, passant de l'état *localisationGood* à *localisationMedium*, puis à *localisationBad*. Dans le premier état, se relocaliser n'est pas nécessaire, dans le second, il est conseillé, et urgent dans le dernier. Une fois relocalisé, l'état de la variable revient à *localisationGood*.

Les objectifs d'enchaînement doivent, eux, être associés à une variable prenant autant de valeur le la longueur de l'enchaînement. Par exemple, pour placer les boîtes jaunes et bleues sur le tapis roulant, l'objectif peut être associé à une variable donnant la couleur de la prochaine boîte à apporter. Cette variable changera d'état lorsque la boîte demandée est déposée sur le tapis roulant.

Explosion de l'espace d'état. Si cette méthode est valide et fonctionne, son inconvénient est de faire exploser l'espace d'état et donc également l'espace de recherche.

Prenons notre exemple pour observer cette explosion. L'état du robot et de l'atelier est représenté par les variables suivantes :

- **robPos** (9 états) : la position du robot.
- **imuPrecision** (2 états) : précision actuelle de la centrale à inertie, *imuAccurate* et *imuInaccurate*.
- **handStatus** (2 états) : le statut du bras du robot, *emptyHand* et *fullHand*.
- **boxColor** (3 états) : la couleur de la boîte tenue par le robot (si il en tiens une) :
  - *noColor* : le robot ne tient aucune boîte, elles sont toute deux en A.
  - *yellow* : le robot tient la boîte jaune, la boîte bleue en A.
  - *blue* : le robot tient la boîte bleue, la boîte jaune en A.
- **chargerStatus** (3 états) : l'état du chargeur, *unknown*, *bad* et *good*.

L'espace d'état est, ici, composé de 162 états. Pour prendre en compte les objectifs proposés et avec la méthode naïve, il est nécessaire d'ajouter les variables suivantes :

- **battery** (2 états) : niveau de la batterie, *batteryBad* et *batteryGood*
- **nextBoxColor** (2 états) : la couleur de la prochaine boîte à apporter, *yellow* et *blue*.
- **avoidAreaRestriction** (2 états) : donne l'état de la zone, *restricted* pour la zone restreinte et *freeAccess* pour la zone libre d'accès.
- **relocaliser** (3 états) : donne la nécessité pour le robot de se relocaliser. Les états se nomment *localisationGood*, *localisationMedium* et *localisationBad*, de la localisation la moins à la plus urgente.

Avec ces nouvelles variables, la taille de l'espace d'état passe à 3888 pour cet exemple trivial. Les méthodes de résolution complètes des MDPs étant très dépendantes de la taille de l'espace d'état (leur complexité temporelle et spatiale est dépendante de la taille de l'espace d'état en $O(S^2)$ [Littman et al., 1995]), le temps ainsi que l'espace nécessaire pour calculer la solution optimale au problème augmentera grandement. Les méthodes complètes ayant besoin d'expliciter le modèle du monde en mémoire, l'espace disponible en mémoire vive risque d'être dépassé, rendant la résolution du problème impossible [Hauskrecht et al., 1998]. Cette explosion sera d'autant plus importante avec la multiplication d'objectifs.

3.3 Hiérarchie de tâches

La technique appelée hiérarchie de tâche [Dietterich, 2000a] (task hierarchy en anglais) permet de contourner le problème de l'explosion du temps de calcul posé par l'utilisation d'états buts. Si cette technique augmente la taille de l'espace d'état comme présenté précédemment, elle ne calcule pas de politique en une seule fois. Elle propose à la place de découper l'objectif en sous-tâches (les sous-tâches pouvant elles-mêmes être redécoupées en sous-sous-tâches, etc.) et de partir des résolutions des sous-tâches élémentaires pour résoudre les tâches d'un rang plus élevé.

3.3.1 Principe

Une sous-tâche st_i est définie comme :

- GS_i : un ensemble d'états buts
- R_i : la fonction de récompense pour st_i
- WS_i : l'ensemble d'états constituant le domaine d'exécution de st_i

La politique π_i résolvant st_i est définie sur l'ensemble d'état WS_i, et son exécution s'arrête lorsque l'état du monde sort de WS_i ou lorsqu'il entre dans l'espace des états buts GS_i. Contrairement à une politique classique, une politique π_i, n'associe pas pour chaque état ws une action a, mais la politique π_j d'une sous-tâche st_j (avec st_j, une sous-tâche de st_i).

Pour être en mesure d'utiliser une sous-tâche j précédemment calculée pour calculer π_i, la méthode calcule la fonction $T(ws, st_j, ws', N)$ [Dietterich, 2000a]. Celle-ci donne la probabilité, en commençant depuis l'état ws, de finir dans l'état ws' après avoir exécuté la politique π_j pendant N pas de temps. Cette fonction requiert la résolution de systèmes d'équations pour être calculée.

Le calcul de la politique π_i s'effectue à l'aide d'une méthode appelée *MAXQ value decomposition* [Dietterich, 2000b], qui dérive de *Value Iteration*. Elle redéfinit les fonctions V_i^* et Q_i^* pour la sous-tâche st_i (avec st_j, sous-tâche de st_i) comme suit :
$V_i^*(ws, i) = 0$ *si* $ws \in GS_i$ *ou si* $ws \notin WS_i$,
$V_i^*(ws, i) = Max_j\ Q_i^*(ws, i, st_j)$, et
$Q_i^*(ws, i, st_j) = \sum_{ws' \in WS} T(ws, st_j, ws', N) \times \gamma^N \times [R_i(ws') + V_i^*(ws', i)]$

Dans le cas des sous-tâches de plus bas niveau, la politique π_i est calculée avec les outils classiques et Value Iteration. Ce calcul permet de calculer une politique globale à partir de sous-politiques avec une approche ascendante.

3.3.2 Limitation

Si cette méthode hiérarchique comporte des avantages en termes de capacité à traiter de plus grands problèmes, elle comporte également plusieurs limites.

Limitation des états buts. Si cette technique permet de modéliser les objectifs en les découpant en sous-tâches de façon récursive, les limitations liées aux états buts subsistent (voir 3.2). En, effet, cette technique ne permet pas d'utiliser toute l'expressivité proposée par les processus Markoviens et ne propose pas de solutions aux objectifs chaînés et de maintien.

États et récompenses intermédiaires. Si la fonction $T(ws, st_j, ws', N)$ permet de connaître l'effet de l'exécution de π_j pour N pas de temps, elle occulte le passage par les états intermédiaires. C'est pour cette raison que lors du calcul de $Q_i^*(ws, i, st_j)$, n'est pris en compte que $R_i(ws')$, c'est-à-dire la récompense au sortir de ces N pas de temps. Les récompenses intermédiaires sont ignorées.

Cela peut être particulièrement gênant dans le cas où une récompense intermédiaire de la sous-tâche st_j ayant une valeur négative (un état à éviter donc) pourrait ne pas être prise en compte dans le calcul de la politique π_i (avec st_j, une sous-tâche de st_i).

Pour pallier ce problème, les sous-tâches doivent être organisées afin que, pour une tâche st_i et sa sous-tâche st_j, que pour tout ws de WS_j, si $R_i(ws) \neq 0$, GS_j inclue ws.

Cela peut cependant rendre compliqué la mise en place de l'architecture en démultipliant le nombre d'états buts définis.

Organisation des sous-tâches. De manière plus générale, la définition et l'organisation des sous-tâches a une très forte influence sur l'efficacité de la politique générée. Son approche uniquement ascendante rend l'algorithme *recursievely optimal* [Kolobov, 2012], puisqu'une tâche j n'a pas de moyen d'influencer une tâche i (avec st_j, une sous-tâche de st_i).

3.3.3 Conclusion

La hiérarchie de tâche est une solution permettant de diminuer le coût des calculs d'un MDP. Pour cela, cette méthode découpe l'objectif en tâches indépendantes et définies sur des domaines réduits, elles-même recoupée en sous-tâches autant de fois que nécessaire. Des politiques sont alors calculées pour résoudre les tâches en partant des tâches les plus élémentaires. Cette méthode tente de calculer une solution à moindre coût en calculant un grand nombre de politiques et de fonctions définies sur des espaces réduits plutôt que de calculer une solution globale "aplatie".

Cependant, cette méthode ne permet pas dans sa forme de répondre aux besoins que nous avons posés. L'utilisation d'états buts nous limite dans la conception d'objectifs complexes. De plus, la hiérarchisation des tâches et leur limitation à un domaine d'état semble délicat à maîtriser. Pour finir, la méthode ne propose pas la possibilité de factoriser le modèle du problème considéré.

3.4 Motivations

Dans une idée similaire aux *drives* (mécanisme permettant de déclencher l'apparition d'un but, voir Section 2.2.4), nous proposons de modéliser les objectifs comme des machines à nombre d'états fini (ou automates). Le but est d'utiliser des automates pour représenter chaque objectif posé à un agent, pour éviter d'ajouter au modèle du monde des variables supplémentaires (voir exemple Section 3.2.1). En utilisant les automates, les objectifs enchaînés, bouclants, ou permanents, pourront être exprimés dans la conception de l'automate plutôt que dans celle du modèle. Avec cette nouvelle méthode de représentation des objectifs, nous développons une structure capable de résoudre des objectifs structurés avec les techniques de décisions Markoviennes classiques.

Idées-forces

- Suivre l'état d'un objectif à l'aide d'une machine à nombre d'état fini, une *motivation* (page 34).
- Introduire la notion de récompense à l'aide des Rewarded Motivation Transitions (page 38).

3.4.1 Machine à nombre d'états fini

Les machines à états sont des structures composées d'états (ou sommet, nœuds, etc.) et de transitions. Les transitions sont des arcs, allant d'un état à un autre, accompagnés d'une condition. Lorsque qu'une transition partant de l'état courant voit sa condition remplie, l'état courant change et prend la valeur de la cible de la transition.

Exemple d'automate. La Figure 3.3 illustre un automate reconnaissant l'expression régulière $aa(ba)^*$. Cet automate possède quatre états, nommés $s0$, $s1$, $s2$ et sf. En commençant dans l'état $s0$ et en lisant les lettres d'un mot les unes après les autres, son état courant évolue. Si la lettre lue correspond à une transition disponible dans l'état courant, la transition se déclenche. Si la lettre lue ne correspond à aucune transition disponible, le processus s'arrête et la reconnaissance de l'expression régulière renvoie un échec. À la fin de la lecture des lettres, si l'état courant est sf, l'expression régulière a été reconnue. Dans le cas contraire, il renvoie un échec.

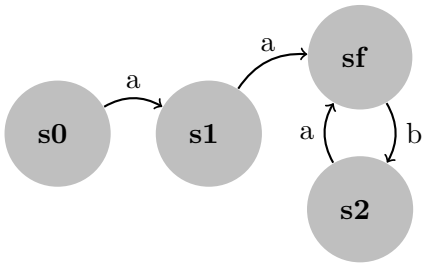


Figure 3.3: Machine à nombre d'état fini reconnaissant l'expression régulière $aa(ba)^*$.

3.4.2 Les motivations, des machines à états particulières

Nous souhaitons à présent utiliser les mécanismes des automates pour représenter un objectif. Modélisés ainsi, nous les appellerons *motivations*. Nous verrons tout d'abord comment suivre passivement l'évolution d'un objectif ainsi modélisé, à l'aide des transitions des motivations. Nous verrons ensuite comment exprimer une notion de récompense en enrichissant ces transitions, devenant des *rewarded motivation transitions*.

Terminologie. Pour éviter les confusions, les états et les transitions des motivations (c'est-à-dire des automates), seront nommés respectivement *motivation state* (ms) et *motivation transition* (mt). Une fois enrichies à l'aide de récompenses, ces dernières deviendront des *rewarded motivation transitions*, abrégées rmt. Noter qu'une liste des termes est présente en début de document, page xi).

Motivation State Vector. Pour décrire l'état motivationnel de l'ensemble des motivations, nous définissons un *motivation state vector* : $msv = (ms1, ms2, ms3...)$ comme un vecteur contenant le *motivation state* de chaque objectif.

Mettre à jour un objectif lors de l'exécution d'un MDP . En utilisant le même mécanisme, que pour l'expression régulière, nous souhaitons utiliser le changement d'état du monde d'un MDP à chaque pas de temps de la même façon que l'automate ci-dessus "lit" chaque lettre. Pour l'instant notre but est uniquement de suivre l'évolution d'un objectif à l'aide d'un automate, appelé *motivation*, en observant l'évolution d'un MDP en temps réel. Pour cela, nous constituons une machine à état dont les conditions des transitions dépendent de l'évolution de l'état d'un problème.

Pour définir les conditions des *motivation transitions*, nous utilisons les *world transitions* wt $[ws, a, ws']$ effectuées à chaque pas de temps dans le MDP. Ainsi, au pas de temps

n, nous considérons la *world transition* (ws, a, ws'), où ws correspond à l'état au pas de temps précédent (ws^{n-1}), a à l'action exécutée (réussie ou non) au pas précédent (a^{n-1}) et ws' au nouvel état courant (ws^n). Le but est donc de définir quelles sont les wt dont l'exécution (voulue on non) implique un changement d'état pour l'objectif, et donc dans l'automate.

À chaque nouveau pas de temps, une fois que l'action a été exécutée (avec succès ou non) et que l'état courant a été mis à jour, nous constituons la *world transition* $(ws^{n-1}, a^{n-1}, ws^n)$, comme illustré Figure 3.4. Nous vérifions ensuite si cette wt remplit la condition d'une *motivation transition* sortant de l'état courant. Si oui, l'état courant de l'automate change en fonction de la *motivation transition* déclenchée.

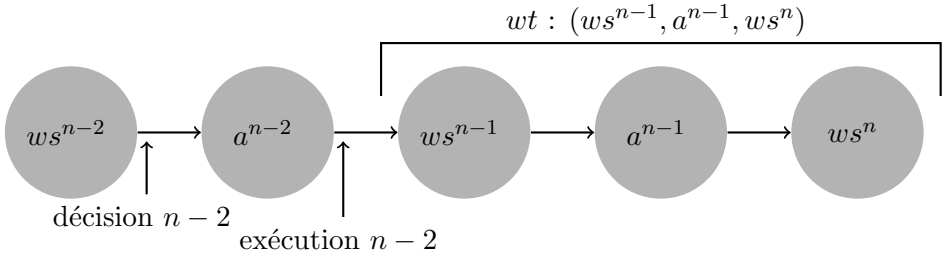


Figure 3.4: Illustration d'une *world transition* dans le déroulement de l'exécution d'un système décisionnel.

Noter que nous ne nous occupons pour l'instant pas de savoir comment sont choisies les actions utilisées dans le MDP, nous observons seulement ses changements d'état pour mettre à jour la situation d'un l'objectif donné. Chacun des trois champs d'une *world transition* peut être remplacé par $*$, signifiant que tout état du monde (ou action jouée) permet d'activer la transition correspondante. Ainsi, la condition $(ws, a, *)$ est remplie lorsque l'action a a été jouée depuis l'état ws, quelle que soit l'état final.

Exemple 1. Pour cet exemple et les suivants, nous utiliserons l'exemple de l'atelier réduit présenté Section 3.1.

Commençons par l'objectif appelé " Alterner sur le tapis roulant les boîtes *jaune* et *bleue*", qui est un objectif enchaîné.

Cherchons à définir les états, ou étapes, qui caractérisent cet objectif. La boîte *jaune* doit être apportée en premier, puis ensuite la boîte *bleue*, puis l'objectif recommence. Pour un découpage plus fin, nous divisons en deux parties chacun des dépôts, il y a donc par exemple les tâches *prendre la boîte jaune* et *déposer la boîte jaune*. Chacune des tâches devant s'accomplir après l'autre, nous avons donc quatre états. Le premier état, *jaune1*, correspond au moment où le robot doit apporter la boîte jaune, mais ne l'a pas encore récupérée. Le deuxième état, *jaune2*, est actif lorsque le robot doit apporter la boîte jaune et qu'il l'a actuellement en sa possession. Les troisième et quatrième états, *bleu1* et *bleue2*, sont les mêmes que le premier et deuxième, mais s'appliquent à la boîte bleue.

Pour constituer un automate dont l'état courant correspondra à l'étape courante de l'objectif, *motivation states*, les *motivation transitions* et leur conditions sont définis pour détecter le passage d'une étape de l'objectif à une autre. Les wt définissant les conditions des mt correspondent alors à des *world transition* significative, dont l'exécution permet de passer d'un état de l'automate à un autre. La transition entre *jaune1* et *jaune2* s'effectue

lorsque le robot réussit à récupérer la boîte jaune, et la transition entre *jaune2* et *bleue1* arrive lorsque la boîte jaune a été déposée avec succès sur le tapis roulant. Les transitions entre *bleu1* et *bleu2* et entre *bleu2* et *jaune1* fonctionnent de manière similaire.

La Figure 3.5 donne la motivation correspondante et qui commence dans l'état *jaune1*. Soulignons encore une fois, qu'il ne s'agit pour l'instant que d'observer l'évolution d'un MDP pour mettre à jour notre objectif; nous n'avons aucun contrôle sur le choix des actions à exécuter.

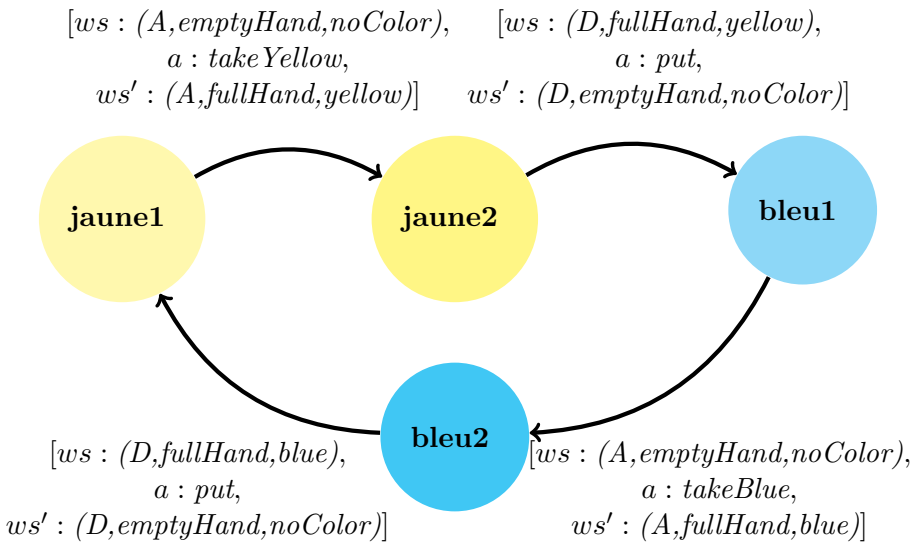


Figure 3.5: Machine à nombre d'états fini dont l'état change en fonction de la progression de l'objectif "Alterner sur le tapis roulant les boîtes *jaune* et *bleue*".
Un état ws ou ws' est ici écrit comme la suite des valuations des variables d'états $(robPos, handStatus, boxColor)$.

Exemple 2. Continuons avec un deuxième exemple, celui de "Maintenir le niveau de la batterie", toujours issu de notre exemple récurrent. Il s'agit d'un objectif de maintien et demande de monter le niveau de la batterie chaque fois que celui-ci devient faible.

Pour cet objectif avons deux états : l'état dans lequel il est nécessaire de recharger la batterie, l'état *faible*, et l'état dans lequel cela n'est pas nécessaire, l'état *bon*. Afin de définir les transitions entre ces deux *motivation states*, nous utilisons le niveau de ressource de la batterie. Lorsque le niveau de la batterie passe sous 50, c'est-à-dire la moitié de l'autonomie du robot, l'état de l'objectif passe de *bon* à *faible*. L'état passe de nouveau à *bon* dès le le niveau repasse au-dessus de ce même seuil. De plus, nous notons une transition bouclante depuis l'état *faible* afin de repérer lorsque le robot se recharge lorsque sa batterie est basse.

La Figure 3.6 illustre ce second automate. Les *world transitions* indiquées contiennent $ws :^* $ et $a :^*$. Ces indications montrent que ws ou a peuvent prendre n'importe quelle valeur pour que la condition soit remplie.

Remarques. Noter que contrairement à l'automate reconnaissant l'expression régulière, nous ne lisons ici plus un mot constitué de lettres, mais un ruban infini de *world transition*

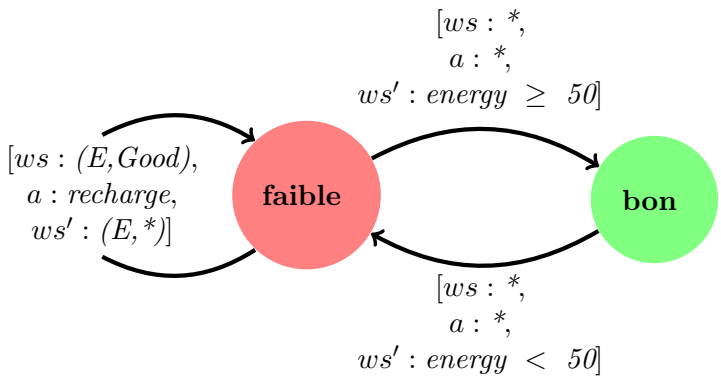


Figure 3.6: Machine à nombre d'état fini dont l'état change en fonction de la progression de l'objectif "Maintenir le niveau de la batterie".
Un état ws ou ws' est ici écrit comme la suite des valuations des variables d'états $(robPos, chargerStatus)$ ou comme une expression dépendant de la ressource *energy*.

$(ws^{n-1}, a^{n-1}, ws^n)$. La lecture de nouveaux états n'ayant pas de fin, les motivations continuent à fonctionner indéfiniment. De plus, pour l'automate reconnaissant l'expression régulière $aa(ba)^*$, si une lettre lue ne pouvait être traitée, l'automate s'arrêtait, la reconnaissance ayant échoué. Dans le cas des machines à état des motivations, les *world transitions* lues qui ne correspondent à aucune transition pouvant être déclenchée sont simplement ignorés.

Suffisant pour modéliser un objectif? Pour le moment, nous avons mis en place le changement de l'état d'un objectif en fonction de l'évolution de l'état du MDP. Nous souhaitons à présent voir si cette structure est suffisante pour guider un mécanisme de décision en lui fournissant des tâches à accomplir.

Nous pouvons constater que pour la motivation de l'exemple 1 "Alterner sur le tapis roulant les boîtes *jaune* et *bleue*", l'automate suit l'évolution de l'objectif, mais contient également les éléments nécessaires pour guider une décision. En effet, la *mt* entre *jaune*1 et *jaune*2 peut être lue comme une tâche : lorsque l'automate est dans le *motivation state* *jaune*1, la tâche à réaliser pour cet objectif est d'être sur la case A et d'y récupérer la boîte jaune.

Cependant, un problème se pose pour l'exemple 2. La *motivation transition* entre *bon* et *faible* est indispensable pour connaître l'état de l'objectif, mais n'est pas une tâche que le robot souhaite accomplir. De plus, même si ce n'est pas visible sur cet exemple jouet, si plusieurs *motivation transitions* sont disponibles, comment choisir celui qui doit être accompli?

Pour remédier à ce problème, nous introduisons des récompenses associées aux *motivation transitions*. Ces *rewarded motivation transitions* permettront de dissocier les transitions à éviter, et celles à accomplir, ainsi que de favoriser une tâche par rapport à une autre.

3.4.3 Un objectif modélisé comme une motivation

Les automates créés pour l'instant sont suffisants pour suivre l'état d'un objectif. Cependant, afin de pouvoir guider le système décisionnel d'un agent à l'aide d'objectifs modélisés à l'aide de motivation, nous enrichissons les *motivation transitions* avec des récompenses. Le but est de pouvoir définir les *world transitions wt* que l'objectif souhaite voir apparaître, puis non seulement de suivre la progression de l'objectif, mais aussi de la quantifier, en calculant la récompense accumulée tout au long d'un expérience.

Rewarded motivation transition. Nous souhaitons pouvoir définir si une *motivation transition mt* doit être accomplie ou évitée, ainsi que son importance. Pour cela, nous changeons les transitions des motivations en *rewarded motivation transitons*, abrégées *rmt*. Comme illustré par la Figure 3.7, les *rmt* sont des *mt* accompagnées d'une valeur de récompense. Une *rmt* sera donc définie comme : $rmt = [(ms, [ws, a, ws'], ms'), r]$.

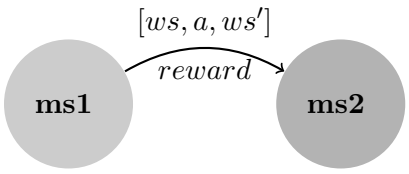


Figure 3.7: Schéma d'une *rewarded motivation transition*.

La notion de récompense des *rmt* fait écho à la fonction de récompense utilisée par un MDP. Nous utiliserons plus tard les *rmt* afin de créer des fonctions de transition sur mesure, permettant de calculer des politiques favorisant la réalisation d'une *rmt* particulière.

Récompense accumulée par une motivation. Pour quantifier les progrès d'une motivation, il nous est désormais possible de cumuler la récompense obtenue par une motivation au cours du temps. Lorsque la condition d'une *rmt* est remplie et que le *motivation state* change, nous ajoutons la récompense r de *rmt* à la récompense accumulée. Il est également possible de mettre à jour R_{sys}, la récompense accumulée par l'ensemble du système, c'est-à-dire la somme des récompenses accumulées par chacune des motivations. Le système de décision que nous développons aura pour but de maximiser le gain de récompense de R_{sys} par unité de temps.

Opérateur utiles pour *rmt*. Pour finir, chaque *rmt* spécifie O_{red}, l'ensemble des opérateurs (ou actions) jugés nécessaires à l'accomplissement de *rmt*. Cet ensemble servira plus tard à diminuer le coût en calcul des politiques visant à accomplir *rmt* en diminuant l'espace d'état des politiques, son utilisation sera discuté Section 4.3.2.

Rewarded world transition. À partir de la définition d'une *rmt*, il est possible d'extraire la *world transition wt* visée et de lui associer la récompense de la *rmt*. Nous formons ainsi une *rewarded world transition* : $rwt = ([ws, a, ws'], r)$. Cette dernière pourra être utilisée comme objectif à atteindre, donnant la transition à exécuter et sa valuation en termes de récompense.

3.4.4 Motivations de l'exemple récurrent

Nous présentons enfin les quatre objectifs de l'exemple récurrent dans les Figures 3.8 à 3.11. Les récompenses indiquées dans les *rewarded motivation transitions* n'ont pas pour but d'être exactes. Les mentions "au bout de X **ut**" sont des conditions sur la ressource du temps. Celles-ci se déclenchent au bout de X **ut** après que la motivation soit entrée dans l'état depuis lequel cette *rmt* sort. Par exemple, si la motivation "Éviter une zone restreinte" est entrée dans le ms "restreinte" avec la valeur $time = 1250$, la condition "au bout de 600 **ut**" se définit comme $[ws : *, a : *, ws' : time > 1850]$ (la ressource de temps donnant le nombre d'unité de temps **ut** depuis que le robot est en fonctionnement).

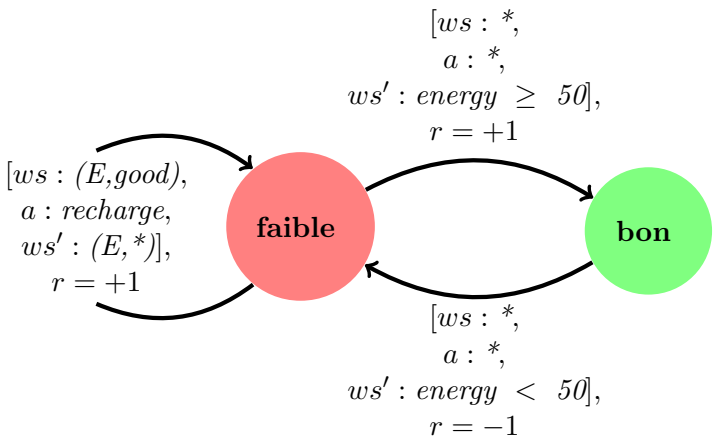


Figure 3.8: Motivation de l'objectif "Maintenir le niveau de la batterie", abrégée **Survie**. Un état ws ou ws' est ici écrit comme la suite des valuations des variables d'états $(robPos, chargerStatus)$ ou comme une expression dépendant de la ressource *energy*.

Pour la motivation "Maintenir le niveau de la batterie", la *rewarded motivation transition* bouclant depuis l'état *faible* permet de favoriser le fait pour le robot de recharger sa batterie. Cependant, l'action *recharge* ne garanti pas que la batterie passe au-dessus du seuil de 50%, le motivation state ne doit donc pas changer. Pour être complète, la motivation intègre la *rmt* entre *faible* et *bon*, qui se déclenche lorsque le niveau de la batterie est effectivement au dessus de 50%.

Exemple de suivi d'une motivation. Pour finir, donnons un exemple de suivi de la motivation "Alterner sur le tapis roulant les boîtes *jaune* et *bleu*". Nous commençons dans l'état du monde $ws = (B,yellowInHand)$, et dans le motivation state $ms = jaune2$.

Encore une fois, nous n'avons pour l'instant aucun contrôle sur les actions exécutées par le robot. Le but est ici de suivre l'état de la motivation en observant la progression de l'état du monde.

ws	a	ms
(B,fullHand,yellow)	moveS	*jaune2*
(F,fullHand,yellow)	moveE	*jaune2*
(G,fullHand,yellow)	moveN	*jaune2*
(C,fullHand,yellow)	moveE	*jaune2*
(D,fullHand,yellow)	put	*jaune2*
(D,emptyHand,noColor)	moveN	*bleu1*
(S,emptyHand,noColor)	moveW	*bleu1*

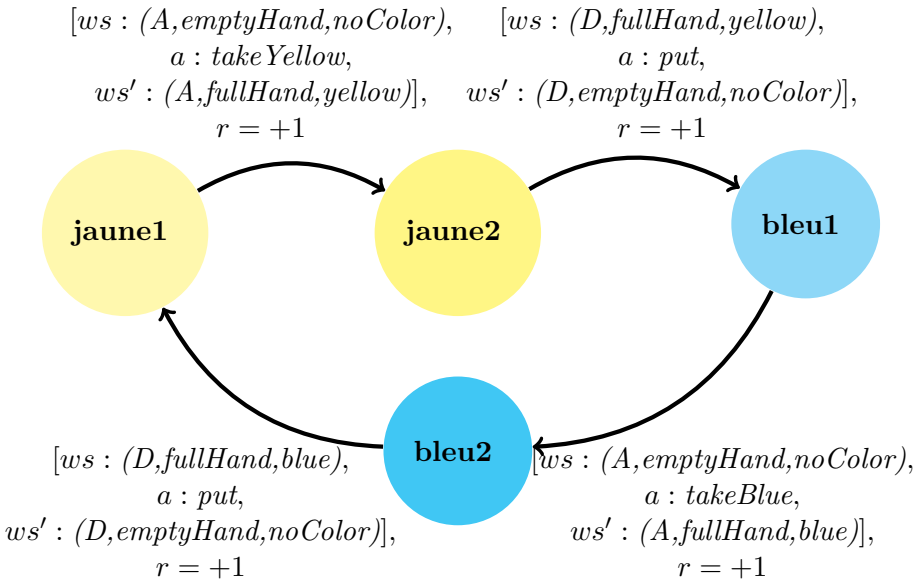


Figure 3.9: Motivation de l'objectif "Alterner sur le tapis roulant les boîtes *jaune* et *bleu*", abrégée **Boîtes**.
Un état ws ou ws' est ici écrit comme la suite des valuations des variables d'états $(robPos, handStatus, boxColor)$.

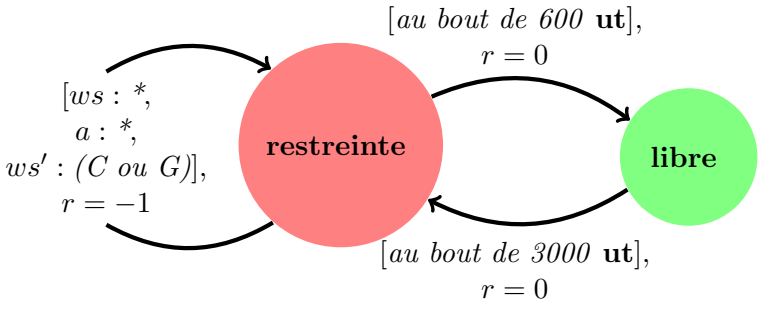


Figure 3.10: Motivation de l'objectif "Éviter une zone restreinte", abrégée **Zone**.
Un état ws ou ws' est ici écrit comme la suite des valuations des variables d'états $(robPos)$.
Plusieurs conditions sont écrites comme des comptes à rebours.

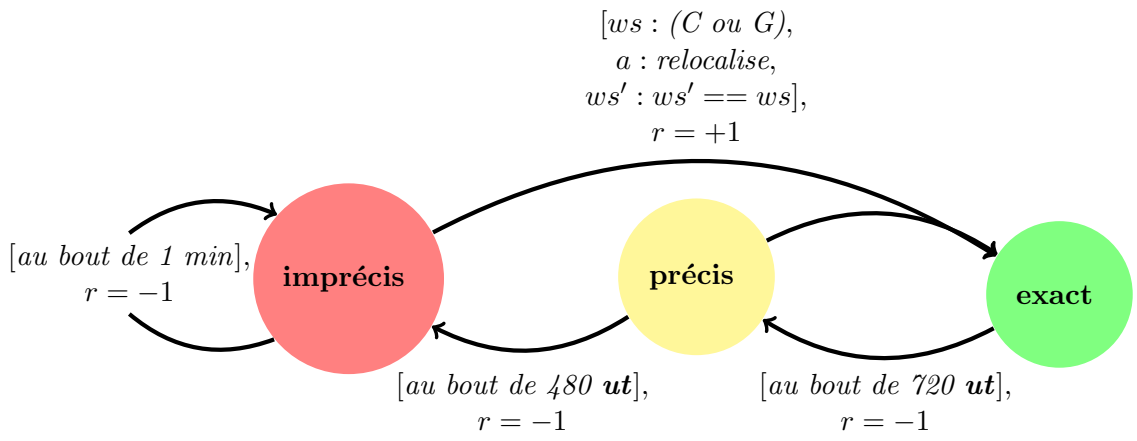


Figure 3.11: Motivation de l'objectif "Se relocaliser", abrégée **Relocaliser**.
Un état ws ou ws' est ici écrit comme la suite des valuations des variables d'états ($robPos$). Plusieurs conditions sont écrites comme des comptes à rebours. Noter que la rmt entre *précis* et *exact* possède la même condition que celle entre *imprécis* et *exact*.

3.5 Conclusion

Dans ce chapitre, nous avons mis en place une nouvelle représentation pour définir les objectifs d'un problème de décision probabiliste, que nous avons nommé *motivation*.

Dans le domaine des MDPs, la manière de représenter les objectifs a une forte incidence sur la résolution d'un problème, en particulier sur la faculté à le hiérarchiser, et la capacité à représenter des objectifs complexes. Les objectifs sont habituellement modélisés à l'aide d'une fonction de récompense unique, adaptée uniquement à la résolution d'un MDP plat. Les états buts (*goal states*) et les techniques hiérarchisées comme *task hierarchy* permettent de découper le problème en sous-niveau. Cependant, leur expressivité reste limitée, peu adaptés pour exprimer les enchaînements de tâches ou les tâches récurrentes.

Les *motivations* sont des objectifs modélisés séparément du modèle du monde sous la forme de machines à nombre d'états fini (ou automate). Une motivation possède un état motivationnel ms et change d'état lorsque des *rewarded motivation transitions* rmt sont déclenchées. Les rmt sont des transitions entre deux états motivationnels et conditionnées par un événement du modèle du monde. Une rmt fait changer d'état la motivation dès que sa condition est remplie, et permet d'obtenir une récompense propre à rmt. La récompense globale du système correspond à la somme des récompenses obtenues par le déclenchement des rmt.

L'expressivité et la structure des motivations nous permet de définir des objectifs complexes ainsi que d'utiliser des méthodes hiérarchiques. Notre objectif est à présent de définir un processus de décision hiérarchisé qui favorise le déclenchement des rmt afin de maximiser la récompense globale.

Chapitre 4

Architecture pour une solution hiérarchique et factorisée

Contents

Dans ce chapitre et pour aborder la résolution de notre problème, nous allons nous intéresser à une version simplifiée : en omettant les ressources. En effet, nous souhaitons traiter les ressources comme des variables d'état à valeur numérique. Ce type de variable n'est pas compatible avec la définition simple des processus de décision Markoviens et nécessite des techniques plus avancées pour être traitées. Ce cas sera l'objet du chapitre suivant.

4.1 Introduction

Dans le chapitre précédent, nous avons proposé une nouvelle méthode afin de représenter des objectifs. Modélisés sous la forme de motivations, ces objectifs vont nous permettre d'effectuer un calcul proche de celui de task hierarchy (voir Section 3.3).

4.1.1 Intuition de départ

Nous nous proposons de :

1. **Casser le modèle MDP en deux** : intégrer des objectifs complexes dans un MDP peut alourdir considérablement le modèle du monde. En considérant un MDP "aplati", c'est-à-dire un MDP constitué tel que le calcul d'une seule et unique politique donne la solution au problème entier, l'introduction d'un objectif de deux tâches enchaînées double la taille de l'espace d'état. Définissons l'objectif enchaîné comme "atteindre l'état $s1$, puis ensuite l'état $s2$". Pour un MDP aplati, il sera nécessaire de doubler la taille de l'espace d'état en ajoutant une variable dont le rôle sera de déterminer si l'objectif en cours est d'atteindre $s1$ ou d'atteindre $s2$. En effet, les objectifs dans les MDPs s'expriment au travers de la fonction de récompense, qui attribue à chaque état une récompense obtenue chaque fois qu'il est atteint. Ainsi, la fonction de récompense ne permet pas de donner une récompense différente pour un état donné, avant et après la réalisation d'un objectif. En conséquence, pour différencier la récompense apportée par un état avant et après la réalisation d'un objectif, nous sommes obligés de dédoubler cet état. Dans le cas de notre objectif enchaîné, la seule option pour exprimer une temporalité, atteindre $s1$ puis atteindre $s2$, est de dédoubler chaque ancien état s pour donner les états $(s, \textit{objectif=s1})$ et $(s, \textit{objectif=s2})$. Ce dédoublement est très fortement redondant, puisque la partie de l'état concernant l'objectif ne change de valeur que pour les transitions $[s, a, s1]$ (avec s et a un état et une action quelconque), qui deviennent $[(s, \textit{objectif=s1}), a, (s1, \textit{objectif=s2})]$ et $[(s, \textit{objectif=s2}), a, (s1, \textit{objectif=s2})]$.

   Cependant, une autre solution consiste à concevoir deux politiques, π_{s1} et π_{s2}, consistant respectivement à atteindre $s1$ et $s2$. Si cette solution permet de s'affranchir du problème de démultiplication de l'espace d'état, elle pose de nouveaux défis que sont l'émergence d'un modèle des objectifs séparé du MDP pour gérer les enchaînements de politiques, et la capacité de prédiction des effets d'une politique dans un tel système. C'est le choix que nous faisons et développons dans ce chapitre.

2. **Instaurer deux modèles séparés : Opérationnel et Intentionnel** : le modèle intentionnel donne le modèle des objectifs, dicte leur comportement et leur progression. C'est également ce modèle qui donne la fonction de récompense du système. Le modèle opérationnel décrit le modèle du monde, c'est-à-dire son espace d'état

ainsi que sa table de transition. Il permet de générer des sous-politiques permettant de résoudre des parties du problème.

3. **Intégrer ces deux modèles au sein d'une architecture** : afin de proposer une solution au problème posé. Cette architecture est composée de trois principaux modules : le *module intentionnel*, le *module opérationnel* et le *module délibératif*. Les deux premiers permettent d'exprimer le problème sous la forme des deux modèles séparés et de calculer des sous-politiques. Le *module délibératif* a pour rôle de gérer les interactions entre les deux modèles et de calculer une solution au problème en modélisant les sous-politiques sous la forme de **macro-actions**, intégrée dans un modèle global formé à partir des modèles intentionnels et opérationnels.

4.1.2 Architecture

L'architecture que nous proposons est composée de trois modules : le *module intentionnel*, le *module opérationnel* et le *module délibératif*. Leurs rôles respectifs et leur interactions sont présentés dans la Figure 4.1.

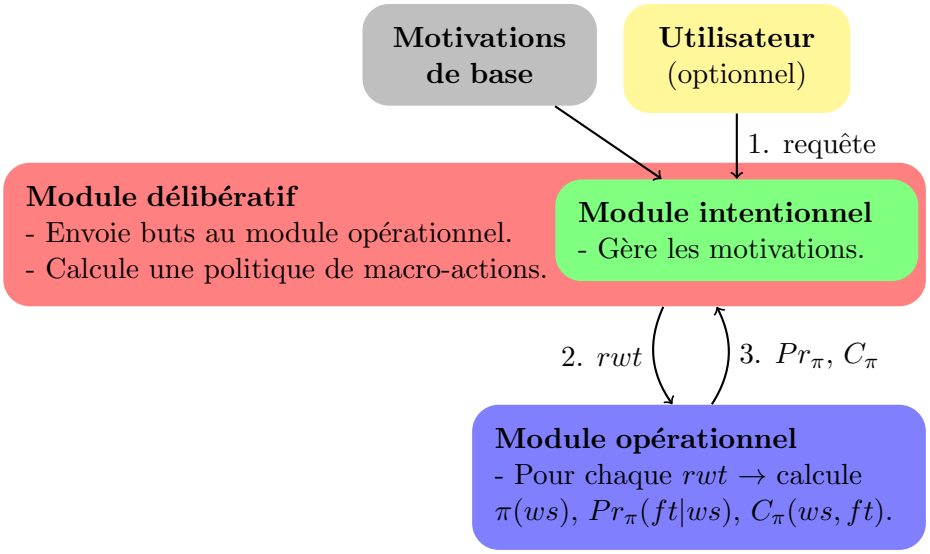


Figure 4.1: Présentation de l'architecture (version sans ressources), composée des modules opérationnel, intentionnel et délibératif.

Motivations de base & **Utilisateur.** Il s'agit de l'ensemble des motivations initialement incluses dans le système. Celle-ci sont pour l'instant écrites à la main. L'utilisateur peut jouer le rôle de pourvoyeur de motivation.

Module Intentionnel. Le module intentionnel (en vert dans la figure) a pour rôle de gérer les motivations, de les tenir à jour et de connaître les changements de *motivation state* possibles. Il définit l'état courant des objectifs, les tâches réalisables, et la manière dont la récompense est obtenue.

Module Opérationnel. Le module opérationnel (en bleu dans la figure) est responsable de la gestion du modèle du monde et de la génération de modèles réduits. Il calcule des politiques pour une tâche donnée, et pour les politiques retournées, détermine leurs effets sur l'état du monde lorsqu'elles sont exécutées (voir définition Section 4.3.4).

Il est composé d'un MDP customisé qui calcule des politiques conçues pour tenter d'atteindre des *rwt* provenant des motivations (voir définition Section 3.4.3) envoyées par le module délibératif. Afin que le module délibératif puisse modéliser l'exécution de ces politiques sous la forme de macro-actions, le module détermine, à partir de chaque *ws* possible, la probabilité de finir sur chacune des transitions buts données par les *rwt*, ainsi que le nombre de pas de temps moyen pour y finir.

Module Délibératif. Le module délibératif (en rouge) est le module principal du processus de décision. Il utilise les données du module intentionnel ainsi que le module opérationnel afin de calculer un ensemble de politiques résolvant des sous-objectifs. Ces politiques ainsi que les données sur leur comportement permettent de construire des macro-actions *ma*, qui modélisent leurs exécution. Pour finir, une politique de macro-actions est calculée, c'est-à-dire une politique qui utilise des macro-actions en lieu et place des actions primitives utilisées habituellement dans les MDPs.

4.2 Module intentionnel

Le module intentionnel est en charge de la définition du problème, défini à l'aide d'un ensemble d'objectifs modélisés sous la forme de motivations. Cette section décrit l'ensemble des éléments qu'il contient, ses fonctionnalités, ainsi que ses relations avec les autres modules.

4.2.1 Éléments et fonctions

Les fonctionnalités du module intentionnel sont entièrement dédiées à la modélisation des objectifs (sous la forme de *motivations*), et la définition des tâches à accomplir à tout moment. De plus, il établit la notion de récompense globale, la valeur que le système cherchera à maximiser.

Motivations. Les motivations permettent d'exprimer les objectifs. Chaque objectif est défini par une machine à nombre d'état fini, définissant un ou plusieurs états possibles pour l'objectif, appelés *motivation states*, ainsi que des transitions, appelés *rewarded motivation transitions* (*rwt*). Les *rwt* sont des transitions entre deux *motivation states* conditionnées par l'exécution d'une *world transition* donnée et octroient une récompense (positive ou non) une fois accomplies. Dès que la condition est exécutée, que cela soit voulu ou non, le changement de motivation state a lieu et la récompense est obtenue. La Section 3.4 est dédiée à la description des motivations.

Donner les *rmt* et *rwt* disponibles. Les *rewarded motivation transitions* sont des transitions entre deux *motivation states* ms et ms'. Une *rwt* se définit comme $([ms, wt, ms'], r)$, et est dite disponible lorsque le *motivation state* courant de la motivation est égal à ms. Une *rewarded world transition* est extraite d'une *rmt* et donne la condition de déclenchement de *rmt* ainsi que la récompense obtenue lorsque la condition est atteinte,

elle se définit comme (wt, r), ou $([ws, a, ws'], r)$. Le module intentionnel donne lorsque demandé la liste des rwt disponibles.

Donner les opérateurs utiles de rmt/rwt. Les rmt/twt définies par les motivations notent, en plus de leur définitions respectives, la liste des opérateurs (ou actions) considérées comme utiles. Il s'agit de l'ensemble des actions qu'il est jugé nécessaire et suffisant pour accomplir une rmt/rwt lorsque celle-ci est disponible. Cette liste d'opérateurs se note O_{red}.

Mettre à jour les motivations. Le module intentionnel a pour fonction de garder à jour l'état des motivations. Pour cela, il vérifie après l'exécution de chaque action si la *world transition* $[ws, a, ws']$ (avec ws l'état précédent, a l'action exécutée et ws' le nouvel état) correspond à la condition d'une rmt actuellement disponible. Dans ce cas, la ou les motivations dont au moins une rmt a été activée changent d'état. De plus, la ou les récompenses correspondantes sont obtenues et ajoutées à la récompense globale.

Établit la récompense globale. La récompense globale du système de décision R_{sys} donne une mesure de son efficacité. R_{sys} est égale à la somme des récompenses obtenues par l'exécution des rmt des motivations.

4.2.2 Communication avec les autres modules

Les communications de ce module ont trois fonctions :

- **Établir / Mettre à jour les motivations** : il s'agit, pendant l'initialisation, d'intégrer les motivations (construites en dehors du système décisionnel) et de définir leurs états de départ. Il est également possible que des requêtes d'utilisateurs changent l'état courant d'une motivation ou changent les récompenses associées aux rmt au cours d'une expérience.
- **Suivre le déroulement des motivations** : à chaque pas de temps, le module déliberationnel envoie au module intentionnel l'évolution de l'état du monde sous la forme d'une *world transition*. Le module intentionnel utilise ces données pour mettre à jour l'état des motivations ainsi que la récompense globale.
- **Donner les rmt disponibles** : lorsque demandé, le module intentionnel transmet au module délibératif la liste des rmt qui peuvent être déclenchées, c'est-à-dire celles pour lesquelles les motivations correspondantes sont dans l'état d'où partent ces rmt.

4.2.3 Exemple

En utilisant notre exemple récurrent (Figure 4.2), nous illustrons le fonctionnement du module intentionnel. Pour cela, nous considérerons la motivation suivante : la motivation **Boîtes**, qui consiste pour le robot à aller chercher alternativement une boîte jaune et une boîte bleue, à l'apporter en D. La Figure 4.3 donne la représentation de la motivation.

Dans cet exemple et en commençant avec la motivation dans l'état *jaune1*, une seule rmt est disponible :

$$(jaune1, [(A, emptyHand, noColor), takeYellow, (D, fullHand, yellow)], jaune2, r = +1)$$

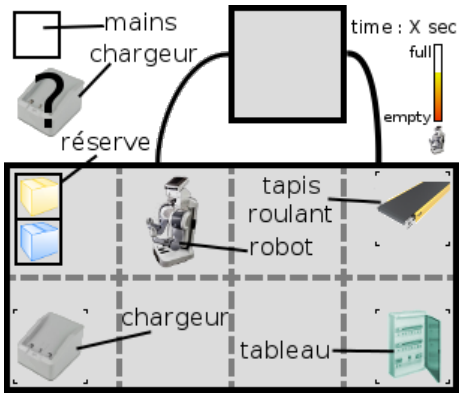


Figure 4.2: Représentation graphique de notre atelier réduit.

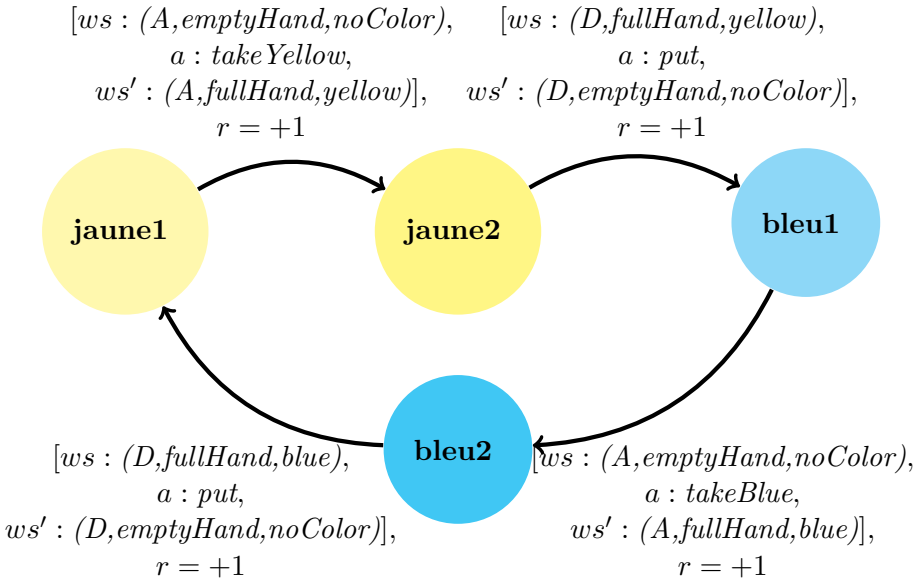


Figure 4.3: Représentation de la motivation **Boîtes**.
Un état ws ou ws' est ici écrit comme la suite des valuations des variables d'état $(robPos, handStatus, boxColor)$.

Pour avoir les détails sur la notation, voir Section 3.4.3. Pour cette *rewarded motivation transition*, la liste des opérateurs nécessaires O_{red} est [*moveN, moveE, moveS, moveW, takeYellow*]. Pour maintenir l'état de cette motivation à jour, il est nécessaire de vérifier, à chaque pas de temps, si une transition [*(A,emptyHand,noColor), takeYellow, (D,fullHand,yellow)*] a été effectuée. Si cette transition vient d'être exécutée, alors l'état de la motivation devient *jaune2*, c'est-à-dire si l'état précédent était *(A,emptyHand,noColor)*, que l'action précédemment jouée était *takeYellow* et que l'état courant est *(D,fullHand,yellow)*. De plus, la récompense globale augmentera de 1.

4.3 Module opérationnel

Le module opérationnel est responsable de la gestion du monde dans lequel évolue notre agent et des outils qui lui permettront d'y effectuer des tâches. Il fournit au module délibératif un moyen pour déclencher les *rmt* données par les motivations : des politiques calculées par un MDP customisé qui favorisent le déclenchement de la condition d'une *rmt*. Nous expliquons ici le moyen choisi pour modéliser le monde, les *Probabilistic STRIPS Operators* (PSO), ainsi que les avantages en termes de représentation que cette méthode offre. Outre la représentation factorisée, les PSOs permettent de définir des sous-modèles, c'est-à-dire des modèles du monde partiels mais exacts. Ceux-ci permettent d'exécuter la plupart des calculs de ce module avec des modèles plus petits, et donc plus rapidement. Une fois établis, le module utilisera ces modèles pour calculer des politiques à l'aide d'un MDP customisé, prenant en argument des ensembles de *rwt* envoyés par le module délibératif (i.e. les transitions buts des *rmt*, plus les récompenses correspondantes). Cela permet de calculer des politiques favorisant l'exécution des *rwt* et donc le déclenchement des *rmt*. Pour finir, nous étudions comment le comportement des politiques calculées peut être prédit. Ces prédictions seront utilisées par le module délibératif pour constituer des macro-actions, qui serviront à calculer une solution au problème.

Idées-forces

- Définir le modèle du monde à l'aide de *Probabilistic STRIPS Operators* (page 49).
- Découper le modèle du monde en sous-modèles (page 53).
- Calculer une politique π_{RWT} pour RWT, un ensemble de *rwt* donné par le module délibératif (page 58).
- Déterminer le comportement de π_{RWT}, la probabilité et le nombre d'actions pour finir par *rwt* afin de permettre au module délibératif de constituer des macro-actions (page 62).

4.3.1 Probabilistic STRIPS Operators

Pour la modélisation du monde dans lequel évolue le robot, nous avons choisi d'utiliser une représentation type *Probabilistic STRIPS Operators* (PSO) [Kushmerick et al., 1995], que nous avons personnalisée. Pour expliquer son fonctionnement, nous verrons dans cette section le formalisme de son ancêtre STRIPS. Nous verrons ensuite comment il a pu être adapté aux modèles probabilistes et comment nous l'utiliserons dans notre système et notre exemple.

Alors que ce chapitre est consacré à une version simplifiée du problème, sans la notion de ressource, nous en tenons compte dans cette partie. Dans le reste de ce chapitre, nous ignorons les coûts en ressources. Les problématiques liées aux ressources seront abordées dans le chapitre suivant.

Formalisme STRIPS

STRIPS est à la base un algorithme nommé Stanford Research Institute Problem Solver [Fikes and Nilsson, 1971]. Il s'agit d'un solveur pour problèmes déterministes. Mais la

partie à laquelle nous nous intéressons est le langage que propose STRIPS pour représenter des problèmes, et en particulier des opérateurs.

Ce langage a inspiré une grande variété de langages de description de problème déterministe, comme PDDL [McDermott et al., 1998] par exemple, et également été dérivé pour traiter des problèmes multi-agents [Kovacs, 2012] ou pour prendre en compte la durée des actions [Fox and Long, 2003].

Exemple d'action STRIPS. La Figure 4.4 illustre une action *put* écrite en langage STRIPS. Ici, l'action utilise deux arguments, O et L (respectivement un objet et une location). Pour exécuter l'action, l'agent doit être en possession d'un objet O ($Has(O)$) et être positionné en L ($Location(L)$), il s'agit des préconditions. Si ces préconditions sont remplies et si l'action est exécutée, les postconditions s'appliquent : le prédicat $Has(O)$ disparaît et le prédicat $Laying(O, L)$ apparaît.

```
Put(O, L)
Preconditions:  Object(O), Has(O), Location(L), At(L)
Postconditions: not Has(O), Laying(O,L)
```

Figure 4.4: Description de l'action *put* en STRIPS.

PSO, une adaptation probabiliste de STRIPS

Le langage STRIPS a été adapté pour convenir au paradigme probabiliste. Cette adaptation porte le nom de Probabilistic STRIPS Operators, abrégé PSO.

Ce langage permet de représenter le modèle d'un problème (ses variables, ses actions, sa fonction de transition) à l'aide de règles. Pour notre travail, nous aurons besoin de connaître le coût en ressources des différentes actions, et ce, selon l'état de départ ainsi que celui d'arrivée. Pour cela, nous avons choisi de développer notre propre version des PSOs, pour pouvoir décrire des actions probabilistes avec les ressources consommées par chaque action.

La Figure 4.5 ci-dessous illustre le modèle de l'opérateur *put* écrit avec un PSO. Son écriture est très différente de la précédente action *put* modélisée avec STRIPS (Figure 4.4). Nous expliquons ci-dessous le fonctionnement de notre description.

```
(action : put
condition variables : rP,hS
effect variables : hS, bC
preconditions : (rP in {'D'}, hS in {'fullHand'})
rules :
        (rP in {'D'}, hS in *) -> ((hS='emptyHand', bc='noColor'), energy= -2 time= +5, 0.85)
        (rP in {'D'}, hS in *) -> ((None), energy= -3 time= +8, 0.15)
)
```

Figure 4.5: Description de l'opérateur *put* à l'aide d'un PSO.

Cette action requiert deux variables d'états pour être décrite, les variables *robotPosition* (rP) et *boxesStatus* (bS). L'action ne peut être exécutée que depuis la case D et avec l'objet dans les mains du robot. Cette action a 85% de chance de succès (la boîte disparaît de ses mains) et 15% de chance de rater (l'objet reste dans les mains). L'exécution de l'action est plus longue et consomme plus d'énergie en cas d'échec qu'en cas de réussite.

Variables d'états et ressources. La première étape pour décrire le modèle du monde d'un problème est la description des variables d'état ainsi que des ressources traitées. Les variables d'état, notées v, sont des variables prenant une valeur parmi un ensemble de valeurs données. Nous traitons des variables avec un nombre libre de valeurs possibles, pas seulement des variables binaires.

L'état du monde est alors défini comme une valuation de chacune de ces variables d'états.

Opérateurs. La seconde et principale partie de la description est celle des opérateurs (i.e. des actions), notés o. Une action se caractérise par un nom unique, et de deux listes, celle des variables dont l'action dépend (c'est-à-dire qui influencent son exécution), nommée liste des variables de dépendance, et la liste des variables dont la valeur peut changer à l'issue de l'action, nommée liste des variables d'effet.

Nous définissons ensuite un ensemble de préconditions. Une **précondition** fournit les valeurs des variables d'état à partir desquelles l'action est réalisable. Pour que l'action puisse être exécutée, au moins une de ces préconditions doit être satisfaite par l'état courant. Une précondition est constituée d'une liste de conditions sur les variables d'état (sélectionnées parmi la liste des variables de dépendance). La **condition** sur une variable peut exiger que la variable ait une valeur précise ou appartenir à un ensemble de valeurs donné. Une précondition s'écrit donc :

condition : $(variable1 : v1, v2, ...) \wedge (variable2 : v3, v4, ...) \wedge ...$
précondition : $(condition1) \vee (condition2) \vee ...$

Pour finir, nous écrivons les règles d'effets des actions, c'est-à-dire l'ensemble des règles qui définissent toutes les transitions possibles entre deux *world states* en utilisant cette action. Une **règle d'effet** est constituée d'une condition et d'un effet qui s'applique à l'état du monde pour définir le nouvel état après l'exécution. La **condition** s'exprime de la même façon que pour une précondition décrite ci-dessus. Les effets sur les variables sont une liste de variables d'état et d'une valeur correspondant à chaque variable. Si l'effet est appliqué, les variables d'état sélectionnées verront leur valeur remplacée par leur nouvelle valeur. Les variables d'état dont les valeurs sont changées doivent faire partie de la liste des variables d'effet. La probabilité correspond à la probabilité que cet effet s'applique. Pour finir, le coût en ressource donne, pour chaque ressource disponible, le coût de l'action pour cette ressource. Un effet se représente comme suit :

règle d'effet : (condition) $\rightarrow$ (effet)
effet : (effets sur les variables, probabilité, coût en ressource)

Note. Noter que si les actions ont des préconditions, celles-ci ne dépendent que des variables d'état et non du niveau des ressources. Les contraintes liées aux ressources seront traitées dans le Chapitre 5.

Génération d'une table de transition. Pour constituer la table de transition pour une action et un état, il suffit de parcourir la liste des effets et de sélectionner les effets dont la condition est remplie. Dans ce cas, les possibles états d'arrivée sont calculés à partir des changements des variables. Ces états d'arrivées constituent la table de transition et

ont une probabilité égale à la probabilité de l'effet. Noter que pour chaque état possible, la somme des probabilités doit être égale à 1.

Un modèle défini à l'aide de PSO est défini comme $WM = (V, O)$, avec :

- V : l'ensemble des variables d'état v. De la forme $\{v1, v2, ...\}$.
- O : l'ensemble des opérateurs o. De la forme $\{o1, o2, ...\}$.

Exemples

Pour illustrer la manière de représenter le modèle d'un MDP avec un modèle PSO, nous utilisons notre exemple récurrent (voir Section 3.1). Nous commençons par donner la définition des variables d'états, puis nous proposons deux actions décrites à l'aide de PSO. Celles-ci montrent comment notre exemple récurrent peut être modélisé, comment il est possible de réduire le modèle d'une action et quelles actions sont difficiles à factoriser.

Les variables d'état sont au nombre de cinq (voir Section 3.2.1) : *robotPosition*, *imuPrecision*, *handStatus* *boxeColor* et *chargerStatus*, abrégées respectivement rP, iP, hS, bC et cS. La Figure 4.6 donne le domaine de chacune des variables. Un état du monde ws peut être par exemple $(B, imuAccurate, fullHand, blue, good)$, c'est-à-dire le robot sur la case B avec une centrale inertielle précise, la boîte bleue dans les mains et le chargeur avec un état fonctionnel.

```
(variables :
        rP : { A, B, C, D, E, F, G, H, S }
        iP : { imuAccurate, imuInaccurate }
        hS : { emptyHand, fullHand }
        bC : { noColor, yellow, blue }
        cS : { unknown, bad, good }
)
```

Figure 4.6: Description des variables d'état du modèle PSO de notre exemple récurrent.

Pour notre premier exemple, nous illustrons l'action *recharge*. Cette action utilise les variables d'état *robotPosition* et *chargerStatus*. Cette action peut être exécutée depuis la case E uniquement et avec la variable *chargerStatus* à *good*, L'action a un taux de succès de 90%, dans ce cas, le robot recharge ses batteries et le chargeur passe à *unknown*. En cas d'échec, la variable *chargerStatus* passe à *bad* et le chargeur doit être réalimenté (voir Section 3.1). La Figure 4.7 illustre comment cette action peut être modélisée simplement à l'aide d'un PSO.

```
(action : recharge
condition variables : rP,cS
effect variables : cS
preconditions : (rP in {'E'}, cS in {'good'})
rules :
        (rP in {'E'}, cS in {'good'}) -> ((cS='unknown'), energy= +300 time= +35, 0.70)
        (rP in {'E'}, cS in {'good'}) -> ((cS='bad'), energy= -1 time= +3, 0.30)
)
```

Figure 4.7: Description de l'action *recharge* à l'aide d'un PSO.

Pour notre deuxième exemple, l'action *moveN*, nous utilisons deux des trois variables d'état de notre exemple récurrent : les variables *robotPosition* et *imuPrecision*. La figure

ne présente que les déplacements partants des cases F et B. Si la centrale inertielle est précise, les déplacements atteignent toujours leur destination. Dans le cas contraire, la règle pour les cases E, F, G et H est : le robot a 70% de chance d'arriver sur la case au nord, 5% sur les autres cases adjacentes et le reste sur la case de départ. Pour les cases B et D, le robot a 50% de chance d'arriver sur la case S, 10% sur chaque autre case adjacente et le reste sur la case de départ. La Figure 4.8 illustre cette action, consistant à se déplacer d'une case vers le haut, décrite à l'aide d'un PSO. Cette action n'étant pas généralisable en utilisant un système de zone pour la localisation du robot, le nombre de règles est important, mais permet cependant d'ignorer toutes les variables non concernées.

```
(action : moveN
condition variables : rP,iP
effect variables : rP
preconditions : (rP in {'B','D','E','F','G','H'})
rules :
        (rP in {'F'}, iP in {'imuAccurate'}) -> ((rP='B'), energy= -2 time= +4, 1.00)
        (rP in {'F'}, iP in {'imuInaccurate'}) -> ((rP='B'), energy= -2 time= +5, 0.70)
        (rP in {'F'}, iP in {'imuInaccurate'}) -> ((rP='E'), energy= -3 time= +8, 0.05)
        (rP in {'F'}, iP in {'imuInaccurate'}) -> ((rP='G'), energy= -3 time= +8, 0.05)
        (rP in {'F'}, iP in {'imuInaccurate'}) -> ((rP='F'), energy= -3 time= +8, 0.20)
        (rP in {'B'}, iP in {'imuAccurate'}) -> ((rP='S'), energy= -4 time= +9, 1.00)
        (rP in {'B'}, iP in {'imuInaccurate'}) -> ((rP='S'), energy= -5 time= +11, 0.50)
        (rP in {'B'}, iP in {'imuInaccurate'}) -> ((rP='A'), energy= -6 time= +14, 0.10)
        (rP in {'B'}, iP in {'imuInaccurate'}) -> ((rP='C'), energy= -6 time= +14, 0.10)
        (rP in {'B'}, iP in {'imuInaccurate'}) -> ((rP='F'), energy= -6 time= +14, 0.10)
        (rP in {'B'}, iP in {'imuInaccurate'}) -> ((rP='B'), energy= -6 time= +14, 0.20)
)
```

Figure 4.8: Description de l'action *moveN* à l'aide d'un PSO.

4.3.2 Sous-modèles

Dans la section précédente, nous avons vu comment représenter le modèle du monde d'un problème en utilisant les PSOs, et comment il est possible de recréer un modèle utilisable pour un MDP. À présent, nous montrons qu'il est également possible de générer des sous-modèles pour MDP à partir du modèle PSO.

Lorsque le modèle du monde est trop vaste pour être exploré, des méthodes locales ou/et des heuristiques sont couramment utilisées pour diminuer l'espace de recherche. Ici, notre but est de traiter de multiples objectifs. Le modèle du monde contient donc toutes les spécifications nécessaires à la réalisation de tous les objectifs auquel notre robot sera confronté. Dans notre exemple récurrent, cela consiste à représenter l'état des boîtes et l'état du chargeur, permettant de calculer une politique à la fois pour recharger la batterie et pour mettre une des boîtes sur le tapis roulant.

Cependant, si nous ne considérons la réalisation que d'un seul objectif à la fois, il y a de fortes chances pour que modèle contienne un certain nombre de variables d'état et/ou d'action n'ayant pas d'influence sur la réalisation du seul objectif considéré. Nous pouvons appeler ces états et actions *indépendantes de l'objectif*, en référence à la notion de *context-specific independance*, qui désigne dans les modèles factorisés le fait que certaines variables d'états n'interviennent dans la résolution d'une action que dans certains cas [Kolobov, 2012] . Pour continuer notre exemple, l'état du chargeur n'influe à aucun moment sur une politique pour amener une boîte sur le tapis roulant, et inversement pour

la variable des boîtes et une politique pour recharger la batterie.

Nous montrons dans cette partie qu'il nous est possible de construire un sous-modèle excluant les états et actions non-spécifique à un objectif. Le sous-modèle produit comporte alors des espaces d'état et d'action réduits tout en restant valides, permettant d'effectuer des calculs rapides par la suite. Ces sous-modèles nous seront utiles pour l'ensemble des calculs permettant de fournir au module délibératif des politiques pour déclencher des *rmt* et de former des macro-actions modélisant l'exécution de ces politiques.

Définition

Nous proposons donc, lorsque nous cherchons à résoudre un unique objectif (déclencher une *rmt*), d'utiliser une version du modèle du monde ne contenant que les états et les actions nécessaires. Nous verrons donc dans cette section comment utiliser notre représentation, utilisant des PSOs, afin de générer un sous-modèle du monde utilisable.

Un sous-modèle est une version d'un modèle du monde réduit pour contenir moins de variables d'état et d'action. À partir du modèle $WM = (V, O)$, nous constituons $WM_{red} = (V_{red}, O_{red})$, où V_{red} et O_{red} sont respectivement des sous-ensembles de V et de O.

Validité

Pour pouvoir utiliser un sous-modèle, il est nécessaire de vérifier si un sous-modèle WM_{red} est *valide*. En effet, puisque les opérateurs dépendent et agissent sur les variables d'états, nous devons vérifier qu'aucun opérateur parmi le sous-ensemble O_{red} ne dépend ou n'agit sur une variable d'état qui ne soit pas présente dans V_{red}. Un opérateur o dépendant ou agissant sur la variable v à besoin de cette variable pour pouvoir générer convenablement un modèle classique. Il est par exemple impossible, pour calculer une politique comprenant une part de navigation, d'utiliser un modèle où la variable de la position de l'agent est absente. De la même façon, si une variable concerne l'état des roues de l'agent (l'état des roues a alors une incidence sur le taux de succès d'un déplacement), cette variable doit être prise en compte.

Définition de la validité. Un sous-modèle valide garantit que, quel que soit o, un opérateur de O_{red}, toutes les variables nécessaires aux conditions de o sont dans O_{red} et qu'aucune variables de $\{V - V_{red}\}$ ne peut changer de valeur à l'issue de son exécution. Pour qu'un sous-modèle soit *valide*, il doit vérifier la formule suivante :

$$WM_{red} = (V_{red}, O_{red})$$
$$\forall o \in O_{red}, (\forall v \in D(o), v \in V_{red})$$

avec $D(o)$, la fonction qui pour un opérateur o renvoie la liste des variables dont l'opérateur dépend ou sur lesquels il agit.

Propriétés d'un sous-modèle valide. Un sous-modèle valide a pour propriété de ne définir qu'une partie du monde, en termes de variable d'état et d'action. Il permet, comme un modèle PSO complet, de générer un ensemble d'états, et une table de transition vérifiant les propriétés du modèle complet.

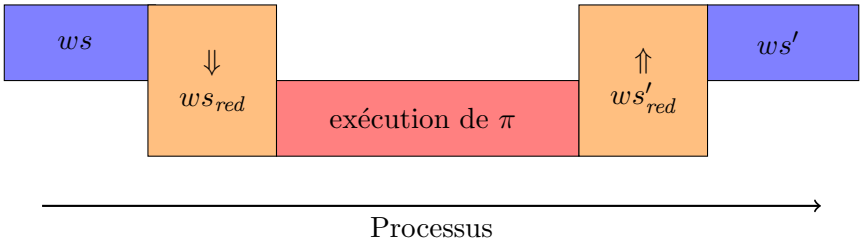


Figure 4.9: Schéma de la réduction.

Puisque chacune des actions (ou opérateurs) peuvent être utilisées pleinement au sein d'un groupe réduit de variable, il est possible d'appliquer une suite d'actions issues de O_{red} aussi bien avec le modèle complet WM, qu'avec WM_{red}. Il est donc possible de rechercher et d'exécuter une politique π au sein de WM_{red} plutôt que dans WM, si toutes les actions présentes dans π appartiennent à O_{red}. Noter que dans ce cas, la politique π s'applique sur un espace d'état réduit.

Il est possible de réduire un état ws pour le faire passer de WM à WM_{red}. Il suffit pour cela de retirer à l'état ws toutes les valuations des variables d'état autre que celles de V_{red}. De plus, l'opération inverse est également possible. Il est possible, après avoir réduit un état ws en ws_{red} et après avoir exécuté des actions, de récupérer ws' à partir de ws'_{red}. Pour cela, la méthode consiste à enregistrer les valuations des variables retirées de l'état ws lors de sa réduction, d'appliquer les actions voulues, puis de rajouter à ws'_{red} les valuations enregistrées pour récupérer ws'. Ce raisonnement est illustré Figure 4.9. Ceci bien sûr à la condition que les opérateurs utilisés pendant la réduction appartiennent tous à O_{red}.

Exemple de sous-modèle

Illustrons à présent l'utilisation des sous-modèles par un exemple. Nous utiliserons ici notre exemple récurrent et un sous-modèle WM_{red} défini comme :

- $V_{red} = \{robotPosition, handStatus, boxesColor\}$
- $O_{red} = \{moveN, moveE, moveS, moveW, put\}$

Dans ce sous-modèle, une politique π est définie et permet au robot de prendre la boîte jaune en A.

Pour commencer, intéressons-nous à la réduction du modèle du monde pour un MDP, en passant du modèle de WS à celui de WS_{red}. Pour la réduction de l'espace d'état, le Tableau 4.1, pour différents états, leur représentation pour ces deux modèles. La valuation de la variable d'état *chargerStatus* disparaît pour ws_{red}, puisque absente du sous-modèle considéré. Nous voyons ensuite la réduction de la table de transition, illustrée Tableau 4.2. Cette table montre six transitions ainsi que la probabilité que ces transition apparaissent, écrite avec le modèle complet, puis ensuite avec le modèle WM_{red}. Noter que dans les deux cas, pour deux écritures complètes différentes, leurs versions réduites peuvent être identiques.

Pour finir, le Tableau 4.3 donne un exemple d'exécution d'une politique définie à partir du sous-modèle valide, et en partant du modèle complet WM. Ici, le sous-modèle WM_{red} est valide, et la politique π a été calculée à partir de celui-ci.

ws	ws_{red}
(E,emptyHand,noColor,unknown)	(E,emptyHand,noColor)
(E,emptyHand,noColor,good)	(E,emptyHand,noColor)
(E,emptyHand,noColor,bad)	(E,emptyHand,noColor)
(E,fullHand,yellow,unknown)	(E,fullHand,yellow)
(A,emptyHand,noColor,unknown)	(A,emptyHand,noColor)
(B,fullHand,blue,unknown)	(B,fullHand,blue)

Table 4.1: Exemple de réduction d'états.
La valeur de la variable *imuPrecision* est *imuInaccurate* pour tous les états.

ws	a	ws'	$T(ws, a, ws')$
modèle complet			
(D,fullHand,yellow,unknown)	put	(D,fullHand,yellow,unknown)	0.15
(D,fullHand,yellow,bad)	put	(D,emptyHand,noColor,bad)	0.85
(E,fullHand,blue,unknown)	moveN	(A,fullHand,blue,unknown)	0.70
(E,emptyHand,noColor,unknown)	moveN	(G,emptyHand,noColor,unknown)	0.05
(B,fullHand,yellow,bad)	moveN	(S,fullHand,yellow,bad)	0.50
(B,fullHand,yellow,unknown)	moveN	(S,fullHand,yellow,unknown)	0.50
sous-modèle WM_{red}			
(D,fullHand,yellow)	put	(D,fullHand,yellow)	0.15
(D,fullHand,yellow)	put	(D,emptyHand,noColor)	0.85
(E,fullHand,blue)	moveN	(A,fullHand,blue)	0.70
(E,emptyHand,noColor)	moveN	(G,emptyHand,noColor)	0.05
(B,fullHand,yellow)	moveN	(S,fullHand,yellow)	0.50
(B,fullHand,yellow)	moveN	(S,fullHand,yellow)	0.50

Table 4.2: Exemple de réduction de la table de transition.
La valeur de la variable *imuPrecision* est *imuInaccurate* pour tous les états.

opération	action	ws - modèle complet	ws - sous-modèle WM_{red}
départ - ws		(G,emptyHand,noColor,unknown)	
réduction		(G,emptyHand,noColor,unknown)	(G,emptyHand,noColor)
$\pi(ws)$	moveN		(F,emptyHand,noColor)
$\pi(ws)$	moveW		(B,emptyHand,noColor)
$\pi(ws)$	moveW		(B,emptyHand,noColor)
$\pi(ws)$	moveW		(A,emptyHand,noColor)
$\pi(ws)$	take		(A,fullHand,yellow)
augmentation		(A,fullHand,yellow,unknown)	(A,fullHand,yellow)
fin - ws'		(A,fullHand,yellow,unknown)	

Table 4.3: Illustration de l'utilisation des sous-modèles pour le processus de réduction - exécution - augmentation.
La valeur de la variable *imuPrecision* est *imuInaccurate* pour tous les états.

4.3.3 Reconstitution d'un modèle classique

La plupart des représentations factorisées, comme les *Binary Decision Diagrams* (BDDs), ou leurs extension, les *Algebraic Decision Diagrams* (ADDs) [Bahar et al., 1997], peuvent être utilisées telles quelles pour calculer des politiques [Boutilier et al., 2000] à l'aide de versions alternatives des algorithmes de décision manipulant directement ces représentations. Des opérations entre ADDs peuvent par exemple être accomplies afin de réaliser la mise à jour de valeur de Value Iteration. Ces méthodes de calcul reposent souvent sur le pari que les gains de la factorisation l'emporteront sur le temps (plus important que pour un modèle classique) nécessaire pour réaliser les opérations.

Contrairement à ces techniques, notre représentation utilisant les PSOs ne sera pas utilisée directement pour calculer une politique. L'objectif de notre représentation tient en la constitution de sous-modèle et à la réalisation de multiples calculs. En ce sens, générer un modèle classique réduit et l'exploiter de nombreuses fois nous sera plus profitable.

Il est possible, à partir d'un modèle type PSO, de reconstituer un modèle plus classique, comme des espaces d'états, d'actions, ainsi qu'une fonction de transition, convenant au formalisme d'un MDP classique.

Nous adoptons une notation utilisée pour les MDP hybrides, en l'adaptant aux coûts fixes de nos actions. Notre modèle est composé des éléments :

- WS : l'ensemble des états du monde. Un état du monde est noté ws et se compose lui-même de deux éléments x et y, $ws = (x, y)$. x comprend la partie de l'état constituée des valuations des variables d'état classiques. y est constitué des valeurs numériques associées aux différentes ressources.

- A : l'ensemble des actions utilisables. Une action est notée a.

- T : la fonction de transition. Puisque les actions agissent sur les niveaux de ressources mais n'en dépendent pas, cette fonction s'exprime uniquement sur la partie x de l'état. $T(x, a, x')$ donne la probabilité depuis un état dont la première partie est x et en exécutant l'action a d'arriver dans un état dont la première partie est x'.

- C : la fonction de coût. Elle permet de connaître la différence du niveau des ressources entre avant et après l'exécution d'une action. $C(x, a, x')$ donne le coût en ressource pour la transition (x, a, x').

Exemple de reconstitution. Nous donnons un exemple de reconstitution d'un modèle classique.

Pour commencer, nous donnons un court extrait de l'espace d'état de notre exemple, illustré Table 4.4, puis un court extrait de la table de transition pour les actions *moveN* et *put*, Table 4.5.

Utilisation des sous-modèles

Le but du module opérationnel est de générer des politiques afin de déclencher des rmt. Or, si un rmt/rwt définit une transition à exécuter, il donne également une liste d'opérateurs à utiliser pour constituer la politique. Cette liste d'opérateurs peut être utilisée pour définir la liste d'opérateur O_{red} d'un sous modèle WM_{red}. En effet, une liste des variables V_{red} minimale peut être déterminée en parcourant les opérateurs o de O_{red} et en ajoutant à V_{red}

état	valuation
1	(E,imuAccurate,emptyHand,noColor,unknown)
2	(E,imuAccurate,emptyHand,noColor,good)
3	(E,imuAccurate,emptyHand,noColor,bad)
4	(E,imuInaccurate,emptyHand,noColor,good)
5	(A,imuAccurate,fullHand,blue,good)
6	(B,imuAccurate,fullHand,yellow,good)

Table 4.4: Extrait de l'espace d'état de l'exemple.

ws	a	ws'	$T(ws, a, ws')$
(D,fullHand,blue,unknown)	put	(D,fullHand,blue,unknown)	0.15
(D,fullHand,blue,unknown)	put	(D,emptyHand,noColor,unknown)	0.85
(D,fullHand,blue,bad)	put	(D,emptyHand,noColor,bad)	0.85
(E,fullHand,yellow,unknown)	moveN	(A,fullHand,yellow,unknown)	0.70
(E,emptyHand,noColor,unknown)	moveN	(G,emptyHand,noColor,unknown)	0.05
(E,fullHand,yellow,good)	moveN	(E,fullHand,yellow,good)	0.20
(B,emptyHand,noColor,bad)	moveN	(S,emptyHand,noColor,bad)	0.50
(B,fullHand,yellow,bad)	moveN	(S,fullHand,yellow,bad)	0.50
(B,emptyHand,noColor,unknown)	moveN	(B,emptyHand,noColor,unknown)	0.20
(B,emptyHand,noColor,bad)	moveN	(C,emptyHand,noColor,bad)	0.10
(E,emptyHand,noColor,good)	recharge	(C,emptyHand,noColor,good)	0.70
(E,fullHand,yellow,good)	recharge	(C,fullHand,yellow,bad)	0.30

Table 4.5: Extrait de la table de transition de l'exemple.
La valeur de la variable *imuPrecision* est *imuInaccurate* pour tous les états.

toutes les variables qui dépendent ou dont la valeur peut être changée par o, garantissant d'obtenir un sous-modèle WM_{red} minimal et *valide*.

Dans le but de calculer des politiques à moindre coût, le module opérationnel peut donc, pour une *rmt*/*rwt* donnée, construire un modèle MDP à partir d'un sous-modèle PSO. Le module peut alors utiliser ce modèle MDP, réduit par rapport au modèle complet, pour calculer une politique avec un espace d'état et d'action réduit. Puisque la validité du sous-modèle est garantie, les politiques calculées pourront être exécutées sur un espace d'état réduit. Il sera également possible d'enchaîner plusieurs politiques s'exprimant sur des sous-modèles différents en augmentant l'état de fin d'une politique avant de le réduire à nouveau pour la politique suivante.

4.3.4 Calcul de la politique optimale pour RWT

Intéressons-nous à présent au calcul d'une politique permettant de favoriser le déclenchement d'une *rewarded motivation transition rmt* envoyée par le module délibératif. Pour cela, nous cherchons à générer une politique dont l'exécution prend fin lorsque la *rewarded world transition rwt* de *rmt* (voir Section 3.4.3) se déclenche. Pour calculer des politiques, nous utiliserons un MDP customisé, ayant la possibilité d'utiliser un sous-

modèle donné. Ce MDP sera utilisé plusieurs fois pour calculer les politiques des différentes *rwt*. Il déterminera lui-même le sous-modèle ainsi que la fonction de récompense à utiliser en fonction de la *rwt* fournie. Ensuite, il modifiera le modèle du monde pour rendre stoppant le fait d'exécuter *rwt* afin d'assurer que la politique calculée s'achève avec la réalisation de l'objectif.

Dans cette section, les modèles du monde utilisés (les états du monde, les actions et la fonction de transition) sont des sous-modèles et correspondent aux *rmt* traités. De plus, rappelons que ce chapitre traite un cas simplifié, sans la notion de ressource. Le coût en ressource des actions est donc ignoré, et nous ne considérons que des motivations ne dépendant pas des ressources.

RWT objectif et fonction de récompense

Notre but est de calculer une politique permettant de déclencher une *rmt* objectif donnée. Pour cela, nous nous servons de la *rmt* donnée pour en extraire la *rwt* objectif correspondante.

$(ms, [ws, a, ws'], ms'), r \rightarrow ([ws, a, ws'], r)$
$rmt \rightarrow rwt$

Définition de RWT. Nous constituons un objectif RWT qui permettra de configurer le MDP customisé. RWT rassemble le sous-modèle WM à utiliser, la *rwt* à exécuter issue de *rmt*, et enfin un nombre libre de *rwt* supplémentaire (qui seront également rendue stoppantes). Les *rewarded world transition* supplémentaires donnent des objectifs secondaires, elles seront détaillées Section 4.4.1. Noter que seule la *rwt* objectif détermine le sous-modèle à utiliser, et non les *rwt* supplémentaires.

$RWT = (WM_{red}, rwt, \{rwt1, rwt2, rwt3...\})$

Fonction de récompense. La fonction de récompense que nous utilisons est définie comme $R : WS \times A \times WS \rightarrow \mathbb{R}$. Chacune des *rwt* de RWT (aussi bien la principale que les supplémentaires) sera utilisée pour définir la fonction de récompense. Une *rewarded world transition*, $rwt = ([ws, a, ws'], r)$, accorde à la transition $[ws, a, ws']$ la récompense r. Toutes les autres transitions auront une récompense nulle. La Figure 4.10 donne l'algorithme permettant de constituer la fonction de récompense. De plus, ces transitions sont déclarées comme finales, signifiant que lorsqu'une de ces transitions est exécutée, l'exécution de la politique prend fin.

Exemple de fonction de récompense. Pour illustrer la constitution d'une fonction de récompense, examinons la constitution de cette fonction pour la politique de la *rmt* (*jaune2*, (*(D,fullHand,yellow)*, *put*, *(A,emptyHand,noColor)*), *bleu1*), $r = +3$ (voir Section 4.2.3). La Table 4.6 donne un extrait de la fonction de récompense.

Customisation du MDP

Pour calculer la politique voulue, nous utilisons un MDP customisé, qui exécutera Value Iteration. Ce MDP prendra donc pour argument RWT et retournera une politique opti-

for all $[ws, a, ws']$ **do**
  $R(ws, a, ws') = 0$
end for
for all $[ws, a, ws']$ **do**
  for all $rwt \in RWT$ **do**
    if $[ws, a, ws'] \in rwt$ **then**
      $R(ws, a, ws') \mathrel{+}= r_{rwt}$
    end if
  end for
end for

Figure 4.10: Algorithme : constitution de la fonction de récompense pour RWT.

ws	a	ws'	$R(ws, a, ws')$
(D,fullHand,yellow)	put	(D,fullHand,yellow)	0
(D,fullHand,yellow)	put	(D,emptyHand,noColor)	3
(E,fullHand,yellow)	moveN	(A,fullHand,yellow)	0
(E,emptyHand,noColor)	moveN	(G,emptyHand,noColor)	0
(B,fullHand,yellow)	moveN	(S,fullHand,yellow)	0

Table 4.6: Exemple de fonction de récompense.

male pour RWT. Cet argument n'étant pas utilisable tel quel par un MDP, nous devons préparer les données pour être utilisées.

La première tâche de la préparation est de générer $< S, A, T, R >$: l'espace d'état, l'espace d'action et la table de transition à partir de la définition de WM_{red} (voir Section 4.3.2). La fonction de récompense est ensuite générée comme expliqué dans la section précédente.

Calculer une politique ayant une fin. La seconde tâche de la préparation consiste à s'assurer que la politique générée correspondra bien à une politique optimale et s'arrêtant lorsqu'une *rwt* est exécutée. En effet, si nous utilisons le modèle actuellement défini, le calcul d'une politique optimale maximisera la récompense obtenue à horizon infini et dans le cas où les transitions des *rwt* peuvent être enchaînées. Or, nous souhaitons qu'une fois une *rwt* exécutée, la politique s'arrête. La raison de ce choix est par exemple que la boîte jaune ne doit être déposée qu'une seule fois sur le tapis roulant dans la motivation "Alterner sur le tapis roulant les boîtes *jaunes* et *bleues*", pour passer de l'état *jaune2* à *bleue1*.

Pour prendre en compte la notion d'arrêt, nous devons empêcher Value Iteration de propager la récompense au-delà des transitions buts définies. Pour cela, nous créons un état virtuel, uniquement présent lors de ce calcul, que nous désignerons à partir d'ici comme le *puits*. Le *puits* est un état à partir duquel aucune action ne peut être exécutée et est utilisé pour piéger la propagation de la récompense. Ensuite, nous dévions les transitions buts gt vers le *puits*, en changeant les transitions $[ws, a, ws']$ en $[ws, a, puits]$. Le *puits* n'a aucune incidence sur la politique retournée, mis à part de garantir l'aspect

while *true* **do**
  update last wt
  if $wt \in RWT$ **then**
    end π *execution*
  end if
  execute $\pi(ws)$
end while

Figure 4.11: Algorithme : exécution d'une politique π_{RWT}

final des transitions buts.

Value Iteration. Une fois le modèle préparé, nous disposons de tous les éléments nécessaires au lancement de l'algorithme Value Iteration :

- l'espace d'état WS : généré à partir du sous-modèle *WM*, auquel s'ajoute le *puits*.
- l'espace d'action A : généré également à partir du sous-modèle *WM*.
- la table de transition T : générée à partir du sous-modèle et modifiée pour s'arrêter à l'exécution des *rwt* de RWT à l'aide du *puits*.
- la fonction de récompense : générée à partir de RWT.

Pour l'exécution de l'algorithme Value Iteration, nous introduisons un marqueur sur chaque état ws qui permettra de savoir si la valeur $V(ws)$ a changé. Cela nous permet de différencier les états pour lesquels il est possible d'exécuter des actions jusqu'à atteindre une *rwt* - et de terminer l'exécution de la politique - et les états pour lesquels aucune issue n'est possible. Lors du processus de décision du module délibératif, cela permettra de savoir si une macro-action *ma* (le modèle d'exécution d'une politique π) peut être exécutée à partir d'un état ou non.

Exécuter une politique π_{RWT}

Une politique π_{RWT} est définie sur un ensemble d'état, en utilisant comme représentation celle d'un sous-modèle. Cette politique est générée grâce à RWT, qui définit l'ensemble des transitions du monde pour lesquelles, une fois atteinte, l'exécution de la politique s'arrête.

Nous définissons l'exécution d'une politique π_{RWT} comme le processus consistant pour un agent à exécuter l'action correspondant à l'état du monde courant et qui se termine lorsqu'une *rwt* de RWT est atteinte. Son exécution est décrite Figure 4.11. Nous définissons la *world transition wt* comme $(ws^{n-1}, a^{n-1}, ws^n)$ pour le pas de temps n. Lorsque cette *wt* correspond à une *rwt* de RWT, le processus s'arrête.

Exemple de politique π_{RWT}. Pour illustrer le calcul de politique, et pour faciliter les exemples suivants, nous donnons ici la politique pour $RWT = \{(C, relocalise, C), +1, (G, relocalise, G), +1\}$, définie dans le sous-modèle WM_{red} suivant :

- $V_{red} = \{robotPosition\}$
- $O_{red} = \{moveN, moveE, moveS, moveW, relocalise\}$

L'action *relocalise* est une action permettant au robot de se recalibrer, elle ne dépend et n'agit que sur la variable d'état *robPosition*, et ne peut s'exécuter que depuis les positions C et G. La Figure 4.12 donne la fonction de transition pour l'action *relocalise* depuis les cases C et G, ainsi que la politique π_{RWT} correspondante, que nous nommerons $\pi_{relocalise}$. Nous considérons dans cet exemple que la variable *imuPrecision* possède la valeur *imuInaccurate*.

Extrait de la fonction de transition pour *relocalise* :

ws	a	ws'	pr
C	relocalise	C	0.50
C	relocalise	B	0.10
C	relocalise	D	0.10
C	relocalise	G	0.30
G	relocalise	G	1.00

Table de la politique $\pi_{relocalise}$:

ws	A	B	C	D	E
$\pi(ws)$	moveE	moveE	relocalise	moveW	moveE
ws	F	G	H	S	
$\pi(ws)$	moveE	relocalise	moveW	moveE	

Figure 4.12: Spécifications de $\pi_{relocalise}$.

4.3.5 Comportement d'une politique π_{RWT} : $Pr(\boldsymbol{rwt}|\boldsymbol{ws}, \pi)$

Une fois la politique optimale permettant d'atteindre une *rmt* calculée, nous nous intéressons au comportement de son exécution, et en particulier à la transition but correspondant à une *rwt* exécutée lors de la fin d'une politique π_{RWT}. En effet, les politiques calculées sont exécutées depuis n'importe quels états pour lesquels la politique renvoie une action et leur exécution se termine lorsqu'une des *rwt* de RWT est atteinte (voir Section 4.3.4). Pour cela, nous modélisons une politique π_{RWT} comme une chaîne de Markov où ne seront gardées que les transitions correspondant à celles de la politique calculée. Il sera ensuite possible de calculer $Pr(rwt|ws, \pi)$ à partir de cette chaîne de Markov. Une fois calculée, il sera donc possible de modéliser l'exécution de π_{RWT} comme une seule action, dont le temps d'action (le nombre d'action exécutées depuis un état de départ pour arriver à la fin de la politique) est variable. Dans cette partie, nous étudions les méthodes permettant de modéliser le comportement d'une politique π_{RWT} afin de pouvoir prédire les effets de son exécution, et pouvoir la modéliser comme une macro-action.

Une politique représentée comme une chaîne de Markov

Les politiques stoppantes s'arrêtent lorsque l'état courant appartient à un ensemble d'états dits *finaux*. Il s'agit là d'un cas plus simple que celui que nous souhaitons traiter, puisque nous utilisons non pas des états finaux, mais des transitions buts $[ws, a, ws']$. Nous nous intéressons cependant à ce cas simplifié afin d'étudier un calcul permettant de connaître $Pr(ws^{final}|ws, \pi)$.

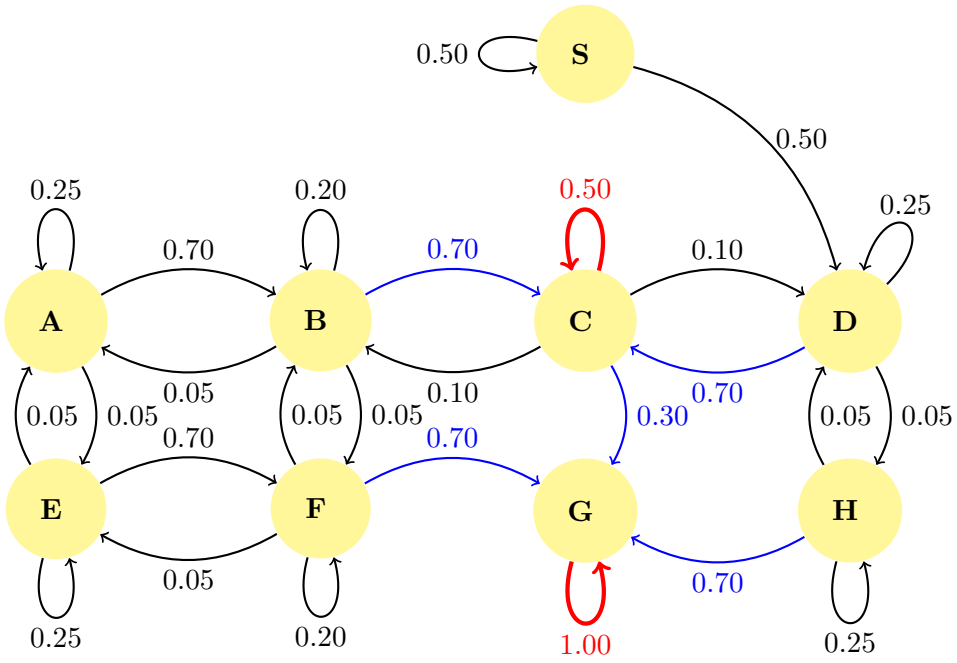


Figure 4.13: Représentation de la politique $\pi_{relocalise}$ comme une chaîne de Markov.

Pour commencer, nous utilisons la table de transition T de notre problème ainsi que la politique π à analyser afin de construire une chaîne de Markov que nous étudierons. La table de transition peut être vue comme une chaîne de Markov où, pour chaque état ws, toute action a réalisable est représentée par un ensemble de transitions vers d'autres états ws', valuées par la probabilité que l'exécution de l'action a change l'état courant ws pour ws'. La chaîne de Markov que nous utilisons est constituée de la même manière, mais où les seules transitions partant de ws sont celles de l'action $\pi(ws)$.

La Figure 4.13 illustre la chaîne de Markov constitué à partir de à la politique $\pi_{relocalise}$ (voir Section 4.3.4). Nous ne considérons que les états pour lesquels la centrale inertielle est imprécise (*imuPrecision=imuInaccurate*). La valeur de cette variable d'état est masquée pour plus de clarté dans la figure et les explications futures.

Systèmes d'équations pour calculer $Pr(\boldsymbol{ws'}|\boldsymbol{ws}, \pi)$

Il est possible à partir de cette chaîne de Markov de calculer $Pr(ws^{final}|ws, \pi)$ en utilisant une technique semblable au calcul de la Policy Iteration. Policy Iteration permet de calculer une politique optimale et a pour particularité, lors de son processus, de calculer la fonction de valeur V en résolvant un système d'équations. Pour cela, il doit résoudre :

$$V_\pi(ws) = R(ws, \pi(ws)) + \gamma \sum_{ws' \in WS} [T(ws, \pi(ws), ws') \times V_\pi(ws')]$$

En modifiant la façon dont ce système d'équation est construit, il est possible de calculer $Pr(ws^{final}|ws, \pi)$:

$Pr(ws^{final}|ws, \pi) = \sum_{ws' \in WS} [T(ws, \pi(ws), ws') \times (Pr(ws^{final}|ws', \pi))]$
avec $Pr(ws^{final}|ws, \pi) = 0$
et $Pr(ws^{final}|ws^{final}, \pi) = 1$

Exemple. Nous utilisons notre exemple précédent, celui de la politique $\pi_{relocalise}$, pour illustrer le calcul de la fonction $Pr(ws'|ws, \pi)$. Pour cela, nous considérons que les états C et G sont finaux (ce qui n'est à l'origine pas le cas) et nous calculerons donc $Pr(C|ws, \pi)$ (respectivement $Pr(G|ws, \pi)$), la probabilité de finir sur la case C (respectivement la case G).

Les Tableaux 4.7 montrent les systèmes d'équations initiaux et une fois résolus (les colonnes non présentées dans le second tableau sont vides).

Représentation d'une équation du système comme une ligne de tableau :

	A	B	D	E	F	H	S	$Pr(C\|ws, \pi)$
B	0.05	0.20	0	0	0.05	0	0	0.70

correspond à (avec $f(ws) = Pr(C|ws, \pi)$) :
$f(B) = 0.05 \times f(A) + 0.20 \times f(B) + 0.05 \times f(F) + 0.70$

Équations initiales :

	A	B	D	E	F	H	S	$Pr(C\|ws, \pi)$
A	0.25	0.70	0	0.05	0	0	0	0
B	0.05	0.20	0	0	0.05	0	0	0.70
D	0	0	0.25	0	0	0.05	0	0.70
E	0.05	0	0	0.25	0.70	0	0	0
F	0	0.05	0	0.05	0.20	0	0	0
H	0	0	0.05	0	0	0.25	0	0
S	0	0	0.50	0	0	0	0.50	0

Après résolution (résultats arrondis) :

ws	$Pr(C\|ws, \pi)$
A	0.88
B	0.93
D	0.94
E	0.12
F	0.07
H	0.06
S	0.94

Table 4.7: Équations pour le calcul de $Pr(C|ws, \pi_{relocalise})$.

Systèmes d'équations pour calculer $Pr(\boldsymbol{rwt}|\boldsymbol{ws}, \pi)$

Le calcul précédent permet de calculer uniquement la probabilité de terminer dans des états finaux.

Pour calculer $Pr(rwt|ws, \pi)$, c'est-à-dire la probabilité de finir sur un seul rwt de RWT, avec $rwt = (ws_{rwt}, a_{rwt}, ws'_{rwt})$, nous résolvons le système suivant :

$Pr(rwt|ws, \pi) = \sum_{ws' \in WS}[T(ws, \pi(ws), ws') \times f(ws, \pi(ws), ws')]$
avec $f(ws, a, ws')$ égal à :

- **pour** $(ws, a, ws') == rwt$: $f(ws, a, ws') = 1$
- **pour** $(ws, a, ws') \in RWT - \{rwt\}$: $f(ws, a, ws') = 0$
- **sinon** : $f(ws, a, ws') = Pr(rwt|ws', \pi)$

Nous présentons également un algorithme commenté permettant de constituer le système d'équations sous la forme d'un tableau Figure 4.14 (voir début de la Tableau 4.7).

lignes : $[ws \in WS_{red}]$
colonnes : $[ws \in WS_{red}]$
tableau : $[lignes][colonnes]$
▷ initialise un tableau représentant un système d'équations
for $ws \in lignes$ **do**
  for $ws' \in colonnes$ **do**
    $tableau[ws][ws'] = T(ws, \pi(ws), ws')$
    ▷ initialise le tableau avec la chaîne de Markov
    if $(ws, \pi(ws), ws') \in RWT$ **then**
      $tableau[ws][ws'] = 0$ ▷ supprime les transitions buts
    end if
  end for
end for
rajouter colonne $Pr(rwt|ws, \pi)$
for $ws \in lignes$ **do**
  if $ws == ws_{rwt}$ *and* $\pi(ws) == a_{rwt}$ **then**
    $tableau[ws][Pr(rwt|ws, \pi)] = T(ws, \pi(ws), ws'_{rwt})$
    ▷ place rwt dans la colonne $Pr(rwt|ws, \pi)$
  end if
end for

Figure 4.14: Algorithme constituant le système d'équation de $Pr(rwt|ws, \pi)$.

Exemple pour une seule rwt**.** Nous réutilisons notre exemple précédent, celui de la politique $\pi_{relocalise}$, pour illustrer le calcul de la fonction $Pr(rwt|ws, \pi)$. Cette fois, aucun état ne sera final, mais puisque $\{(C, relocalise, C), +1, (G, relocalise, G), +1\} \in RWT$, $(C, relocalise, C)$ et $(G, relocalise, G)$ seront des transitions stoppantes. Nous calculerons donc $Pr((C, relocalise, C)|ws, \pi)$ (respectivement $Pr((G, relocalise, G)|ws, \pi)$), la probabilité de finir la politique en effectuant l'action *relocalise* à partir de C et en y restant (respectivement à partir de la case G).

Les Tableaux 4.8 montrent comme précédemment les équations initiales et une fois résolues (de la même façon, les colonnes non présentées sont des colonnes vides).

Équations initiales :

	A	B	C	D	E	F	G	H	S	$Pr(rwt\|ws,\pi)$
A	0.25	0.70	0	0	0.05	0	0	0	0	0
B	0.05	0.20	0.70	0	0	0.05	0	0	0	0
C	0	0.10	0	0.10	0	0	0.30	0	0	0.50
D	0	0	0.70	0.25	0	0	0	0.05	0	0
E	0.05	0	0	0	0.25	0.70	0	0	0	0
F	0	0.05	0	0	0.05	0.20	0.70	0	0	0
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0.05	0	0	0.70	0.25	0	0
S	0	0	0	0.50	0	0	0	0	0.50	0

Après résolution (résultats arrondis) :

ws	$Pr(rwt\|ws,\pi)$
A	0.54
B	0.57
C	0.62
D	0.58
E	0.07
F	0.04
G	0.00
H	0.04
S	0.58

Table 4.8: Équations pour le calcul de $Pr((C, relocalise, C)|ws, \pi_{relocalise})$.
Noter que la variable *imuPrecision* est dans l'état *imuInaccurate* pour tout cet exemple.

Calculer la fonction $Pr(\boldsymbol{rwt}|\boldsymbol{ws},\pi)$ pour tout rwt. Le système d'équation présenté ci-dessus ne concerne le calcul de la fonction $Pr(rwt|ws,\pi)$ que pour un seul rwt donné. Il est cependant possible de calculer cette fonction pour toutes les rwt de RWT. Il suffit pour cela d'ajouter au système d'équations une variable supplémentaire pour tout rwt voulu.

Pour l'exemple précédent, il est possible de rajouter une nouvelle colonne pour le calcul de $Pr((G, relocalise, G)|ws, \pi)$. De la même manière que la colonne pour le calcul de la probabilité de la transition but $(C, relocalise, C)$, la colonne de $Pr((C, relocalise, C)|ws, \pi)$ sera remplie de 0 sauf pour la ligne de G, remplie avec 1.00.

Le rajout d'une colonne supplémentaire ne change pas le temps de résolution du système, il est donc possible de calculer la fonction pour tout rwt sans temps de calcul supplémentaire.

4.3.6 Comportement d'une politique π_{RWT} : $NbAction(rwt, ws, \pi)$

Il nous est à présent possible, en partant d'un état du monde ws, de connaître la probabilité qu'ont les politiques d'atteindre les différentes rwt. Par extension, les rwt étant reliées aux rmt, il est possible de connaître pour une politique π et rwt :

- la probabilité de finir sur rmt
- la récompense gagnée si rwt est atteinte (la récompense de la ou des rmt correspondantes, voir Section 4.3.4)
- l'état courant à la fin de l'exécution de π si rwt est atteinte, et qui est ws'_{rwt}

Cependant, il nous manque un élément afin d'être en mesure de choisir entre l'exécution de deux macro-actions. Si nous connaissons la probabilité ($Pr(rwt|ws, \pi)$) et la récompense obtenue pour les différents états finaux, nous ignorons le nombre d'actions qu'il sera nécessaire d'effectuer afin d'atteindre ces états. En effet, si le nombre de pas de temps pour effectuer une action primitive dans un MDP classique est toujours de 1, une macro-action effectuera plusieurs actions avant de se terminer. La durée d'une macro-action dépend de l'état ws depuis lequel la macro-action est exécutée, et est variable. Nous calculons donc $NbAction(rwt, ws, \pi)$, le nombre d'action moyen nécessaires pour finir spécifiquement par rwt depuis chaque ws. En termes statistiques, il s'agit de déterminer le nombre d'action moyen utilisé en commençant l'exécution de π depuis ws, en ne gardant que les exécutions ayant terminées par rwt et en ignorant les autres.

Pour calculer cette donnée, nous utiliserons une méthode semblable à celle du calcul de $Pr(rwt|ws, \pi)$ de la Section 4.3.5.

Prérequis : $Pr(\boldsymbol{rwt}|\boldsymbol{ws}, \pi)$. Tout comme pour le calcul précédent, le système d'équations à résoudre se base sur la chaîne de Markov de la politique π_{RWT} de la Section 4.13.

Cependant, nous aurons cette fois besoin de la fonction $Pr(rwt|ws, \pi)$ calculée précédemment pour pouvoir constituer le système d'équations. Il est en effet nécessaire de connaître la probabilité de finir par rwt depuis tous les états pour connaître le nombre moyen d'actions lorsque l'exécution de π finit par rwt.

Noter que calculer $NbAction(ws, \pi)$, c'est-à-dire le nombre moyen d'actions pour que la politique π s'arrête, ne requiert pas ce prérequis. Cependant, nous ne nous intéressons pas à cette donnée, car moins précise.

Système d'équations de $NbAction(rwt, ws, \pi)$. Le système d'équations étant plus difficile à former que pour $Pr(rwt|ws, \pi)$, nous ne donnerons pas la formule pour chaque état. L'algorithme présenté Figure 4.15 permet de constituer le système d'équations adéquat (sous la forme d'un tableau, voir Tableau 4.7 pour un exemple d'équation sous forme de tableau).

Exemple. Nous reprenons le calcul de $Pr((C, relocalise, C)|ws, \pi_{relocalise})$ fait lors de l'exemple de la Section 4.3.5. Nous utilisons les résultats de ce calcul pour déterminer $NbAction((C, relocalise, C)|ws, \pi_{relocalise})$. Le Tableau 4.9 illustre la résolution du système d'équation correspondant.

1: $lignes$: $[ws \in WS_{red}]$
2: $colonnes$: $[ws \in WS_{red}]$
3: $tableau$: $[lignes][colonnes]$
4: ▷ initialise un tableau représentant un système d'équations
5: **for** $ws \in lignes$ **do**
6: **for** $ws' \in colonnes$ **do**
7: $tableau[ws][ws'] = T(ws, \pi(ws), ws')$
8: ▷ initialise le tableau avec la chaîne de Markov
9: **end for**
10: **end for**
11: **for** $ws \in lignes$ **do**
12: **for** $ws' \in colonnes$ **do**
13: $tableau[ws][ws'] = tableau[ws][ws'] \times Pr(rwt|ws', \pi)$
14: ▷ pondère la transition par la probabilité, depuis ws' de finir par rwt
15: **if** $(ws, \pi(ws), ws') \in RWT$ **then**
16: $tableau[ws][ws'] = 0$ ▷ supprime les transitions buts du tableau
17: **end if**
18: **end for**
19: **end for**
20: $tableau[ws_{rwt}][ws'_{rwt}] = T(ws_{rwt}, \pi(ws_{rwt}), ws'_{rwt})$
21: ▷ rétablit uniquement la transition de rwt, non pondérée
22: *rajouter colonne* $NbAction(rwt|ws, \pi)$
23: **for** $ws \in lignes$ **do**
24: $total = 0.00$ ▷ compteur des valeurs du tableau
25: **for** $ws' \in colonnes$ **do**
26: $total = total + tableau[ws][ws']$
27: **end for**
28: **if** $total \neq 0$ **then**
29: **for** $ws' \in colonnes$ **do**
30: $tableau[ws][ws'] = tableau[ws][ws']/total$
31: ▷ rétablit la somme des probabilités des transitions de ws
32: **end for**
33: $tableau[ws][NbAction(rwt|ws, \pi)] = 1$
34: ▷ initialise le nombre d'action nécessaire à 1
35: **end if**
36: **end for**
37: $tableau[ws_{rwt}][ws'_{rwt}] = 0$
38: ▷ supprime la transition de rwt

Figure 4.15: Algorithme constituant le système d'équation de $NbAction(rwt|ws, \pi)$.

Équations initiales :

	A	B	C	D	E	F	G	H	S	$NbAction(rwt\|ws,\pi)$
A	0.25	0.74	0	0	0.01	0	0	0	0	1.00
B	0.05	0.20	0.75	0	0	0	0	0	0	1.00
C	0	0.09	0	0.09	0	0	0	0	0	1.00
D	0	0	0.75	0.25	0	0	0	0	0	1.00
E	0.37	0	0	0	0.25	0.38	0	0	0	1.00
F	0	0.71	0	0	0.09	0.20	0	0	0	1.00
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0.75	0	0	0	0.25	0	1.00
S	0	0	0	0.50	0	0	0	0	0.50	1.00

Après résolution (résultats arrondis) :

	$NbAction(rwt\|ws,\pi)$
A	4.34
B	2.98
C	1.55
D	2.89
E	5.78
F	4.55
G	0
H	4.23
S	4.89

Table 4.9: Équations pour le calcul de $NbAction((C, relocalise, C)|ws, \pi_{relocalise})$.

Un calcul de *NbAction*(rwt, ws, π) pour chaque rwt. Contrairement au calcul de $Pr(rwt|ws, \pi)$, il est ici impossible de calculer la fonction pour tous les rwt simultanément. L'impossibilité vient ici du fait que pour calculer $NbAction(rwt, ws, \pi)$, il est nécessaire de modifier le système d'équation spécifiquement en fonction de rwt.

4.3.7 Méthode de résolution rapide pour les systèmes d'équations

Les méthodes présentées précédemment pour calculer $Pr(rwt|ws, \pi)$ et $NbActions(rwt, ws, \pi)$ sont simples, mais peuvent souffrir d'un temps de résolution important lorsque la taille du problème grandit (et ce malgré l'utilisation de sous-modèles). La plupart des méthodes de résolution de système d'équations linéaires avoisinent une complexité de $O(WS^3_{red})$.

Pivot de Gauss

Pour résoudre les systèmes d'équations, nous utiliserons la méthode du Pivot de Gauss, qui s'adapte bien à la nature du problème. En effet, comme le montre les Tableaux 4.8 et 4.9, ces systèmes d'équations ressemblent assez souvent à des matrices creuses, où la plupart des états ne dépendent que d'un nombre restreint d'autres états. Cela est dû au fait que qu'il est rare, dans les MDPs de grande taille, qu'il soit possible en utilisant une action a depuis un état ws de se retrouver dans la totalité des autres états possible du système. De plus, dans les systèmes d'équations que nous résolvons, il est courant d'avoir une ou plusieurs équations très simples, en particulier les equations correspondant aux états dans lesquels il est possible d'accomplir une rwt. Ces équations simples favorisent la résolution des systèmes d'équations grâce à l'algorithme du Pivot de Gauss.

4.3.8 Autre exemple : comportement de $\pi_{recharge}$

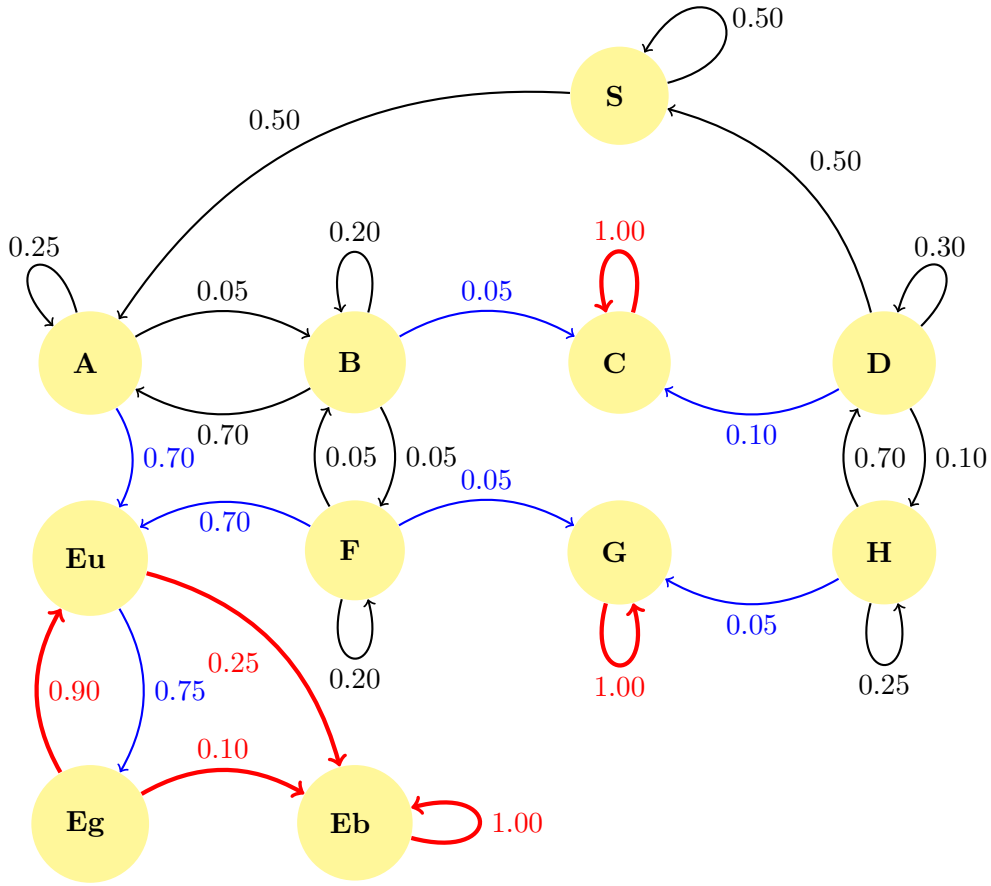


Figure 4.16: Représentation de la politique $\pi_{recharge}$ comme une chaîne de Markov.

Nous introduisons un second exemple pour illustrer nos méthodes déterminant le comportement d'une politique π^{msv}_{rmt}. Nous nous intéressons cette fois-ci à la politique permettant de recharger la batterie du robot. La Figure 4.16 donne la chaîne de Markov correspondant à la politique constituée. Les états Eu, Eg et Eb notent respectivement les états où le robot est en E, et où le chargeur est dans l'état *unknown*, *good* et *bad*. Noter

$rwt1 : (Eu, verify, Eb)$, $rwt2 : (Eg, recharge, Eu)$,
$rwt3 : (Eg, recharge, Eb)$, $rwt4 : (C, wait, C)$, $rwt5 : (G, wait, G)$

ws	$Pr(rwt\|ws, \pi)$				
	$rwt1$	$rwt2$	$rwt3$	$rwt4$	$rwt5$
A	0.25	0.67	0.07	0.00	0.00
B	0.23	0.63	0.07	0.07	0.00
C	0.00	0.00	0.00	1.00	0.00
D	0.19	0.52	0.06	0.22	0.01
Eu	0.25	0.68	0.08	0.00	0.00
Eg	0.00	0.9	0.1	0.00	0.00
Eb	0.18	0.48	0.05	0.21	0.08
F	0.23	0.63	0.07	0.00	0.06
G	0.00	0.00	0.00	0.00	1.00
H	0.18	0.48	0.05	0.21	0.08
S	0.23	0.63	0.07	0.07	0.00

Table 4.10: Résultats du calcul de $Pr(rwt|ws, \pi)$ pour $\pi_{recharge}$.

$rwt1 : (Eu, verify, Eb)$, $rwt2 : (Eg, recharge, Eu)$,
$rwt3 : (Eg, recharge, Eb)$, $rwt4 : (C, wait, C)$, $rwt5 : (G, wait, G)$

ws	$NbAction(rwt\|ws, \pi)$				
	$rwt1$	$rwt2$	$rwt3$	$rwt4$	$rwt5$
A	2.75	3.5	3.5	3.75	5.0
B	3.75	4.75	4.75	2.42	3.67
C	0.00	0.00	0.00	1.0	0.00
D	7.6	8.6	8.6	3.71	4.98
Eu	1.0	2.0	2.0	0.00	0.00
Eg	0.0	1.0	1.0	0.00	0.00
Eb	9.94	10.94	10.94	6.04	4.0
F	2.42	3.42	3.42	3.67	2.26
G	0.00	0.00	0.00	0.00	1.0
H	8.94	9.94	9.94	5.04	3.0
S	5.75	6.75	6.75	4.42	5.67

Table 4.11: Résultats du calcul de $NbAction(rwt|ws, \pi)$ pour $\pi_{recharge}$.
Les transitions épaisses rouges sont des transitions buts, et terminent donc l'exécution de la politique.

que la politique a été simplifiée par endroit, et qu'une partie des états a été masquée. De plus, comme pour l'exemple précédent, la variable *imuPrecision* est dans l'état *imuInaccurate* dans cet exemple, c'est-à-dire que les actions de mouvement peuvent envoyer sur des cases adjacentes ou rester sur place. Son état est masqué pour plus de lisibilité.

Pour cet exemple, nous avons choisi l'état motivationnel $msv = (faible, jaune1, restreinte, précis)$. La politique $\pi_{recharge}$ a été générée pour favoriser le fait de recharger la batterie du robot (la transition (Eg, Eb)). Les transitions buts (Eu, Eb), (Eg, Eb) et (Eb, Eb) permettent de stopper l'exécution de la politique dès que l'état du chargeur passe à *bad*. De plus, les transitions (C, C) et (G, G) sont des transitions buts représentant le fait que le robot est entré dans la zone restreinte. Il existe cinq rwt possible pour ce cas :

- $rwt1$: $([Eu, verify, Eb], -1)$, au moment de vérifier l'état du chargeur, celui-ci a un problème.
- $rwt2$: $([Eg, recharge, Eu], +1)$, le rechargement réussi, l'état retourne à Eu.
- $rwt3$: $([Eg, recharge, Eb], -1)$, le rechargement échoue, l'état passe à Eb.
- $rwt4$: $([C, wait, C], -1)$, le robot va accidentellement sur la case C.
- $rwt5$: $([G, wait, G], -1)$, le robot va accidentellement sur la case G.

L'issue de cette politique est donc très variable, et déterminer son comportement nous permettra de connaître la probabilité des différents scénarios.

Les Tableaux 4.10 et 4.11 illustrent les résultats de nos calculs pour les fonctions $Pr(rwt|ws, \pi)$ et $NbAction(rwt|ws, \pi)$. Cette politique ainsi que ces résultats seront repris dans les parties suivantes.

4.3.9 Synthèse du module opérationnel

Concluons cette partie sur la Figure 4.17, résumant l'action du module opérationnel. Le schéma illustre comment, en partant d'un ensemble de *rewarded world transitions* envoyé par le module délibératif, le module détermine le sous-modèle à utiliser puis calcule la politique optimale à l'aide de ce modèle réduit. Enfin, toujours à l'aide de ce modèle réduit, le module calcule le comportement de la politique calculée précédemment. La politique et les données collectées permettront au module délibératif de concevoir une macro-action pour RWT ma_{RWT}, c'est-à-dire le modèle de l'exécution de la politique π_{RWT}.

4.4 Module délibératif

La fonction du modèle délibératif est de résoudre, partie par partie, les différentes tâches données par les motivations du modèle intentionnel en utilisant le modèle opérationnel, puis d'ordonner ces solutions pour générer une solution au problème global. Cette solution permettra au module de piloter le robot, en déterminant les actions à exécuter, et maximisant la récompense donnée par les motivations, par nombre d'actions jouées. Notre but est de créer une méta-politique, qui donne **lorsqu'une politique se termine**, la nouvelle politique à jouer. Dans cette section, nous expliquons la méthode que nous avons choisie pour pour former RWT, l'argument donné au modèle

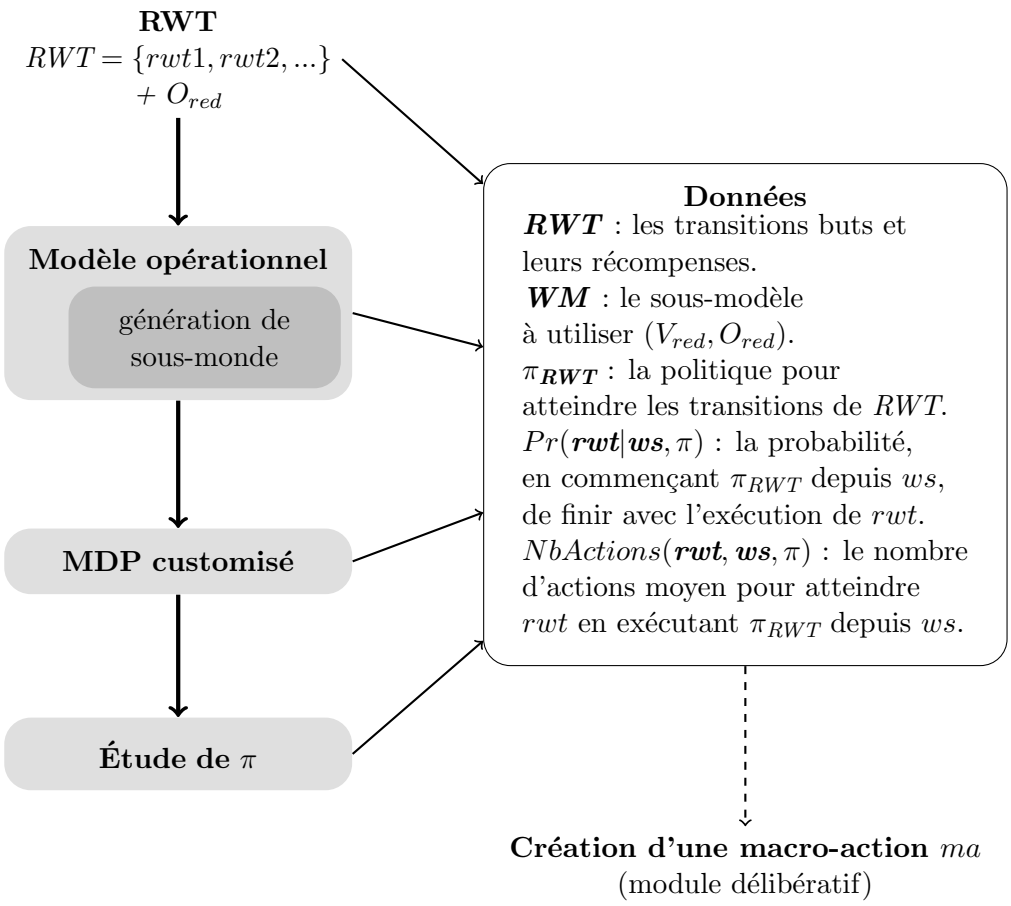


Figure 4.17: Schéma résumant le rôle du module opérationnel dans l'architecture.

opérationnel pour calculer une politique qui favorisera l'exécution de rmt. Ensuite, nous définissons un modèle global, qui décrira la situation du problème en entier, autant sa situation opérationnelle (l'état du monde) que sa situation intentionnelle (l'état des objectifs). Les actions dans ce modèle correspondront aux macro-actions des politiques π_{rmt} jusqu'à ce qu'elles se terminent. Nous extrayons un ensemble d'*états remarquables*, qui correspondent aux états d'arrivées des π_{rmt}. À partir de ce modèle, nous constituons ensuite une méta-politique, définie sur l'ensemble des états remarquables.

Idées-forces

- Générer RWT, afin de calculer une politique pour rmt et en fonction de msv (page 74).
- Constituer un modèle global du problème, avec des macro-actions modélisant l'exécution de politiques π_{rmt} (page 77).
- Dégager un ensemble d'états remarquables (page 79).
- Construire une méta-politique (page 79).

4.4.1 Générer des politiques $\pi^{msv}_{\boldsymbol{rmt}}$

Le but du module délibératif est de maximiserla récompense globale R_{sys} obtenue (voir définition Section 4.2.1) par les motivations. Il doit donc exécuter les actions maximisant l'espérance des gains de récompense. La première étape est donc de déterminer le meilleur moyen pour déclencher efficacement une *rewarded motivation transition* rmt. Pour cela, nous générons des politiques favorisant l'exécution de rmt, à l'aide du module opérationnel.

Cependant, nous ne pouvons pas passer directement une rmt au MDP customisé. Nous devons constituer $RWT = (WM, rwt, \{rwt1, rwt2, ...\})$ (voir définition Section 3.4.3) à partir de rmt. Le sous-modèle et la *rewarded world transition* rwt sont simplement extrait de rmt. Cependant, nous ne pouvons pas laisser le groupe de rwt supplémentaire vide. En effet, prenons le cas où nous cherchons à exécuter $rmt1$, et où $rmt2$ est *disponible*. Si en constituant RWT, nous ne spécifions que WM et $rwt1$ (la rwt extraite de $rmt1$), il y a un risque en exécutant π_{rmt1}, que $rmt2$ soit déclenché pendant l'exécution de la politique. Pour prendre cette incertitude en compte, nous envoyons $rwt2$ dans les rwt supplémentaires de RWT ainsi que toutes les autres rwt correspondant aux rmt disponibles pour un état motivationnel msw donné.

Pour favoriser le déclenchement de rmt depuis msv, les éléments suivant vont constituer RWT, qui sera envoyé en argument au MDP customisé (voir Section 4.3.4) :

- WM : extrait de rmt (voir Section 4.3.3, page 57)
- la rwt principale : extraite de la définition de rmt (voir Section 4.3.4, page 59)
- $\{rwt1, rwt2, ...\}$: la liste des rwt extraites des autres rmt disponibles depuis msv.

Une fois les calculs finis, le module opérationnel renvoie :

- π^{msv}_{rmt} : la politique favorisant l'exécution de rmt
- $Pr(rwt|ws, \pi)$: la fonction qui renvoie, pour chaque rwt possible la probabilité de finir l'exécution de la politique sur la transition de rwt, pour un ws de départ (voir Section 4.3.5)
- $NbAction(rwt, ws, \pi)$: la fonction qui renvoie, pour un ws de départ et une rwt d'arrivée, le nombre moyen d'action jouée pour y parvenir (voir Section 4.3.6).

Gestion des interférences entre les rwt et une politique $\pi^{msv}_{\boldsymbol{rmt}}$. La configuration de RWT présentée ci-dessus pour calculer π^{msv}_{rmt} prend en compte toutes les rmt que la politique pourrait, en le voulant ou par inadvertance, déclencher. Nous voyons ici comment les différentes configurations de RWT et les effets de ces configurations sur le comportement qu'aura la politique retournée.

Trois choix sont possibles :

- Envoyer tous les rwt telles quelles, et constituer un sous-modèle permettant la bonne exécution de chaque rmt. Dans ce cas, il n'existe alors plus qu'une seule politique pour l'état motivationnel msv (elle n'est plus spécifique à une rmt donnée). Il s'agira d'une politique opportuniste, qui choisira automatiquement la rwt à exécuter en fonction d'un compromis entre probabilité, éloignement et récompense promise. Cette méthode a pour avantage de calculer peu de politiques, cependant, il s'agit de politiques calculées sur des espaces de recherches grands. De plus, il n'est plus donné le choix de la rmt à exécuter. Si cette méthode semble parfaite à horizon faible, le comportement de la méta-politique calculée plus tard pourrait être sous-optimale.

- Envoyer tous les *rwt* telles quelles, et conserver uniquement le sous-modèle *WM* de la *rmt* visée à l'origine. Contrairement à la première solution, il peut exister différentes politiques depuis un même msv, en fonction des sous-modèles spécifiés par chaque *rewarded motivation transition rmt*. Il est également possible qu'un ou plusieurs *rwt* ne soient pas atteignable, puisque non définis par le sous-modèle sélectionné. Nous pouvons imaginer une $rwt = ([ws, a, ws'], r)$ dont l'action a ne fait pas partie de *WM*. Le comportement attendu est un comportement opportuniste, mais plus limité que le premier, puisque seule les actions permises par le sous-modèle de la *rmt* visée sont permises. Pour finir, le fait de conserver le sous-modèle original permet au module opérationnel d'effectuer les calculs sur un modèle plus réduit.

- Envoyer normalement la *rwt* de la *rmt* visée, et envoyer les autres en limitant leur récompense promise à un maximum de 0 (les *rmt* dont la récompense est négative ne sont pas affectées). Le sous-modèle choisi pour constituer RWT est celui de la *rmt*. Dans ce cas, il existe une politique différente pour chaque *rmt* à viser depuis l'état motivationnel msv. Ce choix a pour conséquence de prendre en compte chaque *rmt* si ceux-ci sont accessibles. Cependant, le comportement de la politique crée sera dirigée majoritairement vers l'accomplissement de la *rmt* visée au profit des autres et les *rmt* dont la récompense est négative seront évitées. Nous conservons, comme dans le deuxième cas, une taille de l'espace de recherche minimal.

Dans les trois cas, cela permet de prendre efficacement en compte les autres *rmt* et leur probabilité de s'exécuter.

Configuration choisie pour RWT. Si ces trois méthodes ne s'excluent pas, nous avons préféré utiliser la troisième afin de maximiser la possibilité d'accomplir différents objectifs.

Pour générer la politique correspondant à *rmt* depuis msv, nous envoyons donc les éléments suivants :

- *WM* : issu de *rmt*

- la *rwt*, dite *principale* : issue de *rmt*, non modifiée

- l'ensemble de *rwt* supplémentaires : constitué des *rwt* issues des autres *rmt* disponibles pour msv, en modifiant leurs récompenses. Les récompenses négatives sont conservées, permettant de garder son influence dans la politique créée. Les récompenses positives sont passées à 0, ce qui permet à la politique retournée par le module opérationnel d'être attirée en priorité par la *rwt* principale, tout en étant ne négligeant pas les *rwt* négatives.

Notons que si RWT est constitué à partir du couple (rmt, msv) (la *rewarded motivation transition rmt* et de l'état motivationnel msv), il est fort probable qu'il existe un couple (rmt, msv') pour lequel nous constituons un RWT identique. Dans ce cas, Les politiques, les prédictions sur le comportement, ainsi que plus tard le modèle des transitions globales, ne seront calculées qu'une seule fois.

Exemple de constitution de RWT

Pour donner un exemple, nous constituons un exemple d'objectif à partir de notre exemple récurrent.

L'état motivationnel *msv* à partir duquel nous partons est *(faible,bleu2,restreinte,précis)*, ce qui correspond aux motivations dans les états suivants :

- Maintenir le niveau de la batterie = *faible*
- Alterner sur le tapis roulant les boîtes jaune et bleue = *bleu2*
- Éviter une zone restreinte = *restreinte*
- Se relocaliser = *précis*

Les *rmt* disponibles depuis *msv* que nous prendrons en considération sont : (les *rmt* dépendant des ressources, abordées dans le Chapitre 5, seront ici ignorées)

- Maintenir le niveau de la batterie
  $\rightarrow (ws$: *(E,Good)*$, a$: *recharge*$, ws'$: ***$)$
- Alterner sur le tapis roulant les boîtes jaune et bleue = *bleu2*
  $\rightarrow (ws$: *(D,blueInHand)*$, a$: *put*$, ws'$: *(D,emptyHand)*$)$
- Éviter une zone restreinte = *restreinte*
  $\rightarrow (ws$: ***$, a$: ***$, ws'$: *(C)*$)$
  $\rightarrow (ws$: ***$, a$: ***$, ws'$: *(G)*$)$
- Se relocaliser = *précis* $\rightarrow$ aucune

La *rmt* que nous voulons déclencher est celle qui consiste à exécuter une fois l'action *recharge* depuis la case E, de la motivation **Maintenir le niveau de la batterie**. Pour constituer RWT, nous nous intéressons tout d'abord à la *rmt* principale et au sous-modèle WM_{red} qu'elle défini. Dans notre cas, nous avons :

$$WM_{red} = (V_{red} = \{robotPosition, chargerStatus\},$$
$$O_{red} = \{moveN, moveE, moveS, moveW, verify, power, recharge\})$$

D'après le sous-modèle que nous avons choisi, nous pouvons retirer la *rmt* de *bleu2* : il est en effet impossible de la déclencher, puisque l'action *put* est indisponible. Il ne reste donc que deux *rmt* : la principale, et celle de la motivation "Éviter une zone restreinte". La RWT que nous enverrons au module opérationnel sera alors :

- $WM_{red} = (V_{red} = \{robotPosition, chargerStatus\},$
  $O_{red} = \{moveN, moveE, moveS, moveW, verify, power, recharge\})$
- *rmt* **principale**
  $\rightarrow (ws$: *(E,Good)*$, a$: *recharge*$, ws'$: ***$)$ (Maintenir le niveau de la batterie)
- *rmt* **supplémentaires**
  $\rightarrow \{(ws$: ***$, a$: ***$, ws'$: *(C ou G)*$)\}$ (Éviter une zone restreinte)

4.4.2 Modèle global

Afin de traiter le problème dans son ensemble, nous construisons un modèle global issu des modèles intentionnels et opérationnel.

Un état global est défini comme (ws, msv), un couple constitué d'un état du monde ws et d'un vecteur des états motivationnels msv. Un état global se note gs.

$$gs = (ws, msv)$$

Modélisation des politiques comme des macro-actions. À présent que la notion d'état est définie, intéressons-nous aux actions disponibles dans ce modèle.

Les politiques π_{rmt}^{msv} calculées précédemment ont pour effet de favoriser des changements d'états motivationnels en se terminant par l'exécution d'une rmt. Nous utilisons ces politiques pour passer d'un état gs à gs'. L'exécution d'une de ces politiques forme alors une macro-action ma, et sera utilisée dans notre modèle global de la même manière qu'une action primitive l'est dans un MDP classique.

$ma_\pi \rightarrow$ modélisation de l'exécution de π pour le modèle global

Notons que depuis l'état global (ws, msv), seules les macro-actions modélisant des politiques π_{rmt}^{msv}, et avec $ms \in msv$, seront utilisables.

Transition globale. Maintenant que nous avons défini les états et les actions du modèle global du module délibératif, nous étudions la modélisation des transitions globales (gs, ma, gs') (avec $ma \rightarrow \pi_{rmt}^{msv}$), et de la fonction de transition qui donnera $Pr(gs'|ma, gs)$.

Nous étudions donc les effets d'une politique sur un état global gs, afin de construire un modèle de transition permettant de déterminer $gs' = (ws', msv')$, l'état global atteint à la fin de l'exécution de ma, et la probabilité d'atteindre gs'. Pour cela, nous utiliserons la fonction $Pr(rwt|ws, \pi)$ (calculée Section 4.3.5). Pour cela, nous cherchons parmi les rwt de RWT celles pour qui la probabilité est non-nulle depuis gs.

Connaître les rwt dont l'exécution est possible nous permet alors de déterminer les *world states* ws' possibles. Cela nous permet également de déterminer msv', à l'aide des motivations, en examinant quelles sont la ou les rmt déclenchées par l'exécution de rwt. Le choix pour lequel nous avons opté pour la constitution de RWT (voir Section 4.4.1) garanti que rwt seul est suffisant pour connaître l'ensemble des rmt déclenchées. Nous pouvons donc déterminer des transitions (gs, π, gs') possible et connaître leur probabilités, qui est $Pr(rwt|ws, \pi)$.

Fonction de récompense globale. Une fois les transitions entre *global state* établies, il est possible d'établir une fonction de récompense pour notre modèle globale. Les transitions (gs, ma, gs') permettent de connaître les rmt déclenchées entre gs et gs'. Nous pouvons donc associer à chacune des transitions globales la somme des récompenses promises par les rmt déclenchées.

Exemple de modèle global

Nous utilisons l'exemple récurrent et développé tout au long de cette thèse pour illustrer la constitution d'un modèle global. Cet exemple s'appuie sur la politique $\pi_{recharge}$ développé Section 4.3.8.

Commençons par donner la conception de quelques états globaux $gs = (ws, msv)$. Le tableau suivant donne quelques exemples d'états que nous réutiliserons dans les exemples suivants.

gs	ws			msv			
	rP	bS	cS	survie	boîtes	zone	relocaliser
$gs1$	D	emptyHand	unknown	faible	jaune1	restreinte	précis
$gs2$	D	blueInHand	unknwon	faible	bleu2	restreinte	précis
$gs3$	E	emptyHand	unknown	bon	jaune1	restreinte	précis
$gs4$	E	emptyHand	bad	faible	jaune1	restreinte	précis
$gs5$	C	emptyHand	unknown	faible	jaune1	restreinte	précis

Pour continuer notre exemple, nous utilisons comme macro-action la politique $ma \rightarrow \pi_{ex} = \pi^{msv1}_{rmt1}$, avec :

- $msv1 = (faible, jaune1, restreinte, précis)$
  $msv2 = (bon, jaune1, restreinte, précis)$
- $rmt1 = (msv1, (*, recharge, (E, unknwon)), msv2), +1$

Cette macro-action correspond à la politique consistant à charger la batterie une fois et alors que la zone des cases C et G est actuellement restreinte. Nous considérerons que la situation actuelle est telle que l'exécution de $rmt1$ aura pour effet de changer le *motivation state* de la motivation "Maintenir le niveau de la batterie" pour l'état *bon*.

Donnons à présent les transitions, leur probabilités de se produire et leur récompense associée en exécutant la politique π_{ex} depuis $gs1$. Le tableau suivant donne pour chaque transition globale possible la probabilité qu'elle se produise, le nombre d'action moyen pour la déclencher (la colonne nommée Γ), ainsi que la récompense obtenue à l'issue de l'exécution dans chaque cas.

transition globale	pr	Γ	R
$(gs1, \pi_{ex}, gs3)$	0.52	8.6	+1
$(gs1, \pi_{ex}, gs4)$	0.25	7.84	0
$(gs1, \pi_{ex}, gs5)$	0.22	3.71	-1

Noter que les valeurs des colonnes de pr et de Γ sont arrondies. De plus, les valeurs de la ligne $(gs1, \pi_{ex}, gs4)$ correspondent à l'agrégation de deux cas : lorsque l'action *verify* annonce que l'état du chargeur est *bad*, et lorsque l'action *recharge* rate, changeant également l'état du chargeur pour *bad*. Notons enfin que les résultats en partant de l'état $gs2$ sont similaires à ceux de $gs1$, en remplaçant la valeur de la motivation "Alterner sur le tapis roulant les boîtes *jaune* et *bleue*" par *bleu2* pour les états globaux $gs3$, $gs4$ et $gs5$.

Sélection des états remarquables

Le nombre d'état globaux possible est égal à la multiplication de l'espace d'état WS et celui de l'espace des vecteurs des états des motivations MSV. Cela signifie que le nombre

d'état globaux peut exploser si un problème de taille non-trivial possède un grand nombre de motivations.

Cependant, afin de calculer une méta-politique à partir d'un état global gs, il n'est pas nécessaire de disposer de tous les états globaux. En effet, le nombre d'états globaux dans lesquels il est possible de se trouver à l'issue de l'exécution d'une politique π_{rmt}^{msv} est limité. Cela est dû au fait que le nombre d'état finaux (c'est-à-dire l'état ws^n situé à la fin de rwt) pour une politique est restreint.

Prenons par exemple la politique renvoyée par la RWT définie ci-dessus. Les états ws dans lesquels il est possible de finir sont :
(E,Good), *(C,Good)*, *(C,Bad)*, *(C,Unknown)*, *(G,Good)*, *(G,Bad)* et *(G,Unknown)*.
Il s'agit de tous les états possibles à la fin des deux rmt de RWT.

De plus, certain *world state* ws ne peuvent apparaître qu'associés à un *motivation state* ms précis. Par exemple, le *world state (A,blueInHand)* (l'état final de la rmt disponible depuis *bleu1*) ne peut apparaître que dans le cas où l'état de la motivation "Alterner sur le tapis roulant les boîtes jaune et bleu" est dans l'état *bleu2*. De ce fait, hormis une situation de départ qui peut être quelconque, si toutes les politiques π_{rmt}^{msv} retournées par le module opérationnel sont exécutées jusqu'à leur fin, l'état global gs à la fin de l'exécution d'une politique appartient à un ensemble réduit d'état globaux appelé *états globaux remarquables*.

Une fois toutes les politiques π_{rmt}^{msv} connues et une fois leur fonction $Pr(rwt|ws, \pi)$ calculée, il est possible de déterminer l'ensemble des *états globaux remarquables*. Pour générer l'ensemble des état finaux remarquables : faire pour chaque msv possible et pour chaque rmt disponible depuis msv, pour tous les rwt pour lesquelles $Pr(rwt|ws, \pi) \notin O$ (avec ws quelconque), ajouter l'état global (ws^{rwt}, msv').

Constituer un modèle global pour les états remarquables. Nous pouvons à présent constituer un modèle complet pour les états remarquables du système.

Les politiques pour les couples (msv, rmt) connues et leur comportement prédit, nous pouvons générer un sous-ensemble d'états constitués des états finaux possibles des politiques. Il est ensuite possible de générer toute les transitions possibles entre ces états, connaissant les prédictions, ainsi que la fonction de récompense associée.

4.4.3 Calcul d'une méta-politique des états remarquables π^r

Une fois le modèle global mis en place pour les états remarquables, nous adaptons les méthodes classiques des MDPs pour calculer une politique optimale à l'aide d'un algorithme Value-Iteration modifié. Pour prendre en compte la durée d'exécution des macro-actions (c'est-à-dire le nombre moyen d'actions nécessaires à sa réalisation), le facteur γ sera porté à une puissance égale au nombre moyen d'action nécessaire pour l'exécution. Le calcul de la mise à jour de la fonction de valeur devient alors la suivante :

$$V^{r*}(gs) = Max_{ma \in \Pi} \sum_{gs' \in GS} \gamma^{\Gamma(gs,ma,gs')} \times T(gs, ma, gs') \times [R(gs, ma, gs') + V^{r*}(gs')]$$

Avec Π, l'ensemble des macro-actions disponibles depuis gs.
$\Gamma(gs, ma, gs')$, la fonction donnant le nombre d'actions moyen à exécuter en partant de ws pour arriver à rwt en exécutant la macro-action ma.

La politique calculée se nomme méta-politique π^r et associe des couples (gs, ma),

donnant la macro-action ma à exécuter depuis le *global state* gs. afin de maximiser la récompense obtenue par les motivations.

Cette méta-politique est partielle, puisque ne prenant pas en compte tous les états gs possibles. Cependant, elle garantie de donner la macro-action optimale après l'exécution d'une macro-action quelconque ou si l'état global de départ fait partie des états remarquables. La constitution des états remarquables permet de donner les macro-actions à exécuter pour tous les états rencontrés plus tard après l'exécution de chaque macro-action.

Pour les calculs futurs, concernant les états autres que remarquables, la fonction de valeur V^{r*} sera conservée à l'issue de ce calcul. Celle-ci donne la récompense à long terme obtenue depuis chaque état.

4.4.4 Calcul d'une méta-politique globale π^g

Le précédent calcul de la méta-politique garantie de donner la politique optimale à exécuter depuis les états globaux remarquables, c'est-à-dire l'ensemble des états globaux obtenus pour l'exécution de toutes les macro-actions et pour tout état global possible. Cependant, elle ne donne pas la politique à jouer pour les autres états, et en particulier celui d'un état global initial arbitraire.

Calcul la politique à exécuter depuis un autre état global. Puisque la méta-politique calculée donne la macro-action ma à exécuter pour chaque état global après l'exécution d'une seule macro-action, la seule difficulté, pour un autre état global, est donc de choisir la première politique à exécuter. Il n'est nécessaire, pour un état global arbitraire gs, que de calculer la meilleure politique à exécuter depuis cet état. Pour cela, nous prenons toutes les rmt disponibles depuis cet état, et générer les transitions globale correspondantes, ainsi que les récompenses obtenues pour ces transitions. Une fois calculés, nous déterminons la macro-action optimale en utilisant la fonction de valeur précédemment calculée (Section 4.4.3).

$$\pi^g(gs) = Maxarg_{ma \in \Pi} \sum_{gs' \in GS} \gamma^{\Gamma(gs,ma,gs')} \times T(gs, ma, gs') \times [R(gs, ma, gs') + V^{r*}(gs')]$$

Avec Π, l'ensemble des macro-actions disponibles depuis gs.

$\Gamma(gs, ma, gs')$, la fonction donnant le nombre d'actions moyen à exécuter en partant de ws pour arriver à rwt en exécutant la macro-action ma. Et V^{r*}, la fonction de valeur calculée pour la méta-politique π^r.

Nous obtenons alors la macro-action ma optimale à jouer depuis cet état global. La méta-politique π^r peut ensuite être utilisée normalement.

Calcul d'une méta-politique π^g pour tout gs. Il est possible, en utilisant le calcul précédent, d'ajouter une entrée à la méta-politique pour chaque états global gs. Cela permet de créer une méta-politique optimale, pour tous les états globaux possibles et avec moins de calculs que dans le cas où la méta-politique est calculée pour tous les états dès le départ. En effet, séparer les états remarquables des autres permet de calculer normalement π^r pour les états remarquables, et en une seule itération de valeur les autres états.

1: **while** *Vrai* **do** ▷ exécution de la méta-politique
2: $gs \leftarrow$ *état courant*
3: **if** $gs \notin \pi^r$ **then**
4: *calculer* $\pi^r(gs)$
5: **end if**
6: $\pi = \pi^r(gs)$
7: **while** *Vrai* **do** ▷ exécution d'une macro-action
8: $ws \leftarrow$ *état du monde courant*
9: *exécuter l'action* $\pi(ws)$
10: **if** *rwt atteinte* **then**
11: *break*
12: **end if**
13: **end while**
14: **end while**

Figure 4.18: Algorithme pour l'exécution de π^r.

4.4.5 Analyse de π^r et π^g

Les deux méta-politiques que nous proposons, la méta-politique pour les état remarquables π^r et la méta-politique globale π^g, ont des caractéristiques différentes. Nous analysons ici leurs différences en termes de méthode d'exécution et d'optimalité.

Analyse de la méta-politique pour les états remarquables π^r

Cette méthode propose, pour un problème donné, de ne garder qu'un sous-ensemble d'états remarquables et de naviguer de l'un à l'autre en exécutant des macro-actions. Une macro-action représente l'exécution d'une politique visant la résolution d'un sous-problème, par la réalisation d'un changement d'état dans une des motivations, et entraînant l'obtention de récompense (positive ou négative). La méta-politique π^r cherche alors à optimiser les gains de récompenses à long-terme en choisissant les macro-actions dont l'espérance combinée de la récompense à court et à long terme est la meilleure.

Une méta-politique π^r s'exécute en deux strates, comme le montre la Figure 4.18. Les lignes $3-5$ correspondent au calcul de la première macro-action si l'état global gs initial ne fait pas partie des états remarquables. L'exécution commence par déterminer l'état global actuel et la macro-action à exécuter, puis entre dans une phase où la politique de la macro-action sélectionnée s'exécute jusqu'à la fin (voir définition d'une politique stoppante Section 4.3.4). Il n'est pas possible, dans le cas de cette méta-politique, de changer de macro-action une fois commencée, car les macro-actions pour les états intermédiaires ne sont pas calculées.

Cette méthode propose de résoudre le problème en découpant l'espace des états en un sous-espace des états que nous qualifions de remarquables. De plus, la temporalité de l'exécution de la solution est segmentée entre l'exécution de la méta-politique et l'exécution des macro-actions. Ces deux segmentations nous amènent à parler, pour cette solution, d'*optimalité hiérarchique* [Kolobov, 2012]. Cette notion définit le fait d'utiliser de façon optimale des sous-solutions, elles-mêmes optimales pour leur sous-problème. Cependant, le fait de hiérarchiser le problème et de segmenter l'exécution en résolution de sous-problèmes affaibli la notion d'optimalité de la solution.

1: **while** *Vrai* **do**
2: $gs \leftarrow$ *état courant*
3: *exécuter* $\pi^g(gs)(ws)$ ▷ exécute la première action de la politique de gs
4: **end while**

Figure 4.19: Algorithme pour l'exécution de π^g.

Analyse de la méta-politique globale π^g

Cette seconde méthode propose, à partir de la méta-politique π^r, de calculer une méta-politique π^g, pour tous les états globaux gs possibles. Pour cela, elle calcule pour tous les états non-remarquables la meilleure macro-action à exécuter maximisant la récompense obtenue à long terme.

Si nous exécutons π^g comme nous exécutons π^r, seuls les états remarquables seront sollicités après l'exécution de la première macro-action. Tous les états globaux gs pour lesquels nous avons calculés $\pi^g(gs)$ sont alors inutiles. Cependant, nous pouvons changer de stratégie et opter pour l'exécution d'uniquement la première action de la macro-action retournée par π^g, comme montré Figure 4.19. De cette manière, l'algorithme reconsidère après chaque action jouée quelle est la prochaine à exécuter.

Si cette seconde méthode nécessite de calculer la macro-action à exécuter depuis chaque état global (même si nous ne choisissons d'en exécuter que la première action), elle permet d'éviter la segmentation temporelle imposée par la méthode précédente. Cependant, la hiérarchie due au découpage en sous-tâche persiste, et même si la méta-politique π^g permet donner une action pour chaque état global possible, cette méta-politique calcule bel et bien des macro-actions à exécuter. De ce fait, cette seconde méthode propose une méta-politique de même optimalité, mais dont l'exécution sera plus efficace grâce au fait que nous ne sommes plus obligé d'attendre la fin de l'exécution d'une macro-action avant d'en sélectionner une nouvelle. Cela peut en outre avoir une incidence pour les problèmes dans lesquels l'agent peut dévier involontairement de sa route, en lui permettant de reconsidérer son prochain objectif au cours de son exécution.

Conclusion de l'analyse

Si la méta-politique π^g semble être plus efficace, elle nécessite, après avoir calculé π^r, d'effectuer un calcul, certes simple, pour un nombre d'états globaux qui pourrait être très important. Il est toutefois intéressant de noter que si ce calcul risque d'être long, la complexité spatiale, elle, demeurera faible. Cela est dû au fait que seule la méta-politique π^r ainsi que la fonction de valeur V^{r*} sont nécessaires aux calculs.

Il pourrait également être intéressant de tester un facteur détectant dans quel état global il est intéressant de calculer une nouvelle macro-action à exécuter. Cette solution permettrait de changer de macro-action lorsque sa progression se passe mal, tout en ne conservant qu'un nombre minimal d'entrée à π^r.

Remarquons que le calcul de π^r ou de π^g pourrait servir d'initialisation pour le calcul d'une politique complète à l'aide de techniques classiques. Cela aurait pour effet de poser les bases d'une politique où la propagation de la récompense espérée à très long terme a déjà été effectuée. Dans ce cas, il serait possible que la fonction de valeur calculée converge rapidement. Cependant, le calcul de politique complète sur de grands espaces

reste limité par la complexité spatiale, à cause de la nécessité d'expliciter l'ensemble des états [Hauskrecht et al., 1998].

4.5 Conclusion

Dans ce chapitre, nous avons proposé une solution à un problème de décision probabiliste avec des objectifs multiples et permanent. Les objectifs sont constitués comme des enchaînements de sous-objectifs et modélisés à l'aide de *motivations* (présentés Chapitre 3). Nous omettons la problématique liée aux ressources dans ce chapitre.

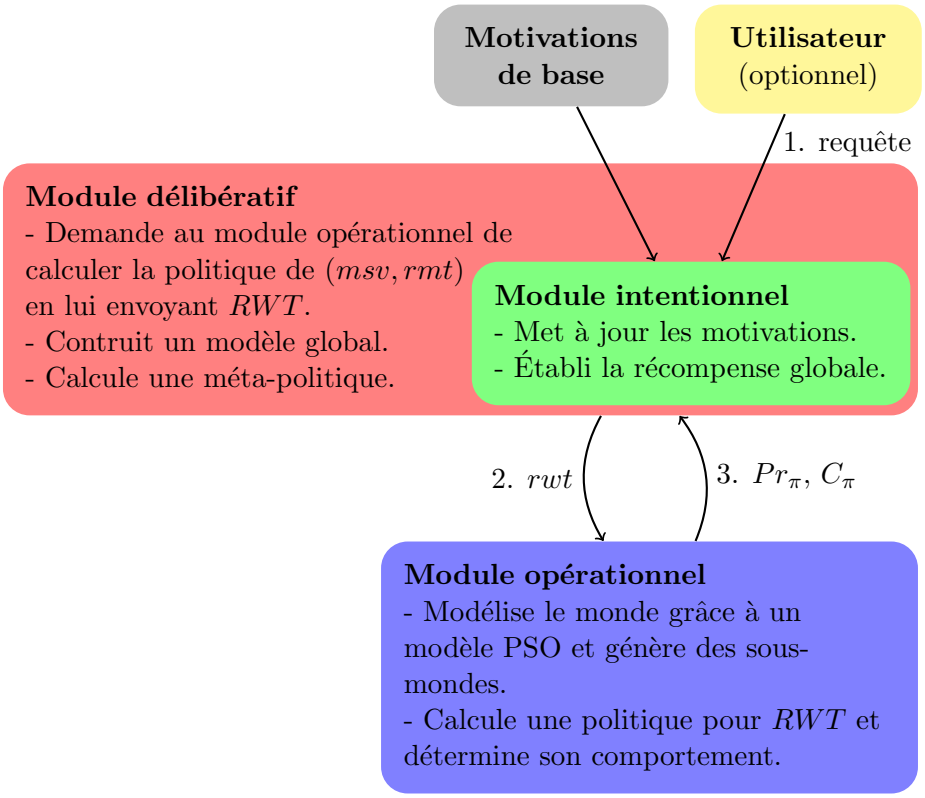


Figure 4.20: Architecture complète (version sans ressources), composée des modules opérationnel, intentionnel et délibératif.

Pour résoudre ce problème, nous avons proposé une architecture capable de trouver des solutions aux sous-objectifs, de modéliser des macro-actions à partir des ces solutions. Les macro-actions s'intègrent alors dans un modèle global permettant de calculer une solution sous la forme d'une politique utilisant les macro-actions calculées.

L'architecture, résumée Figure 4.20, est composée de trois modules : intentionnel, opérationnel et délibératif. Le module intentionnel permet de centraliser les motivations, donne les sous-objectifs à atteindre pour une situation donnée et établi la notion de récompense du système, obtenue pour la résolution de sous-objectifs. Le module opérationnel modélise le le monde à l'aide d'un modèle PSO permettant de définir des sous-modèles du monde. Il permet de calculer des politiques pour des sous-objectifs et de déterminer le comportement des politiques retournées. Le module délibératif est chargé de constituer un modèle global en réunissant le modèle du monde et les motivations.

Il utilise ce modèle pour commander au module opérationnel les politiques à calculer, puis utilise les politiques et leur prédictions afin de modéliser des macro-actions, qui deviennent les actions primitives de ce modèle global. Enfin, en déterminant un ensemble d'états remarquables le module calcule une méta-politique pour le problème, qui maximise la récompense obtenue par la résolution des sous-objectifs.

Chapitre 5

Architecture pour une solution hiérarchique et factorisée avec ressources

Contents

Nous allons dans ce chapitre aborder la thématique des ressources et comment les traiter comme des variables continues, afin de traiter notre problème de décision, avec de multiple objectifs, de multiples ressources et visant l'autonomie du système quant au choix des tâches à réaliser et de l'ordre dans lequel les réaliser.

5.1 Introduction

Le chapitre 4 nous a permis de trouver une solution à un problème modélisé à partir d'un modèle opérationnel (un modèle PSO décrivant un MDP) et d'un modèle intentionnel (un ensemble de *motivations*).

La résolution du problème se fait alors en hiérarchisant la solution, en calculant des politiques résolvant les *rewarded motivation transitions* (rwt) des motivations puis générant des macro-actions, modélisant l'exécution de ces politiques. Un modèle global du problème est alors défini à partir duquel une politique de macro-actions est calculée. La solution apportée peut être appliquée sans limite de temps, le système étant décrit dans son ensemble.

Si cette solution convenait précédemment, elle se heurte à un certain nombre d'obstacles lorsque la problématique des ressources est posée. Deux types de ressources seront considérées : le temps et l'énergie (le niveau de la batterie). L'incorporation de ces ressources comme des variables numériques **continues** est rendue difficile en particulier pour la ressource temps. Il ne s'agit pas ici seulement d'une ressource que l'on intègre afin de chercher une solution maximisant la récompense accumulée en fonction du temps écoulé. Le temps joue un rôle plus important puisque les motivations peuvent intégrer des dates limites par exemple.

5.1.1 Intuition de départ

Dans l'impossibilité de ramener le problème avec ressources au problème sans ressources, nous allons adapter la solution développée dans les chapitres précédents pour en tenir compte. Nous choisissons de ne pas intégrer les variables continues des ressources lors du calcul des politiques π^{msv}_{rmt}, en considérant que le niveau des ressources n'a pas d'influence sur la fonction de transition (leur niveau étant toutefois modifié lorsqu'une action est exécutée). Cependant, nous ne pouvons pas ignorer l'importance du niveau des ressources lors du calcul d'une méta politique. Nous choisissons donc de calculer les consommations en ressources des macro-actions issues des π^{msv}_{rmt}, puis de calculer une méta politique locale (limitée dans le temps) à partir d'un état donné. La durée de l'expérimentation n'étant pas bornée, l'exécution alternera alors le calcul d'une nouvelle méta politique locale et son exécution. Nous proposons donc de :

- **Calculer le coût en ressources des macro-actions** : en utilisant les techniques présentées Section 4.3.6, nous allons calculer les coûts moyens en ressources des politiques calculées pour déclencher des *rewarded motivation transitions*.
- **Définir un cycle délibération / exécution** : la méthode du chapitre précédent pour calculer une méta politique complète ne peut plus être appliquée en utilisant des ressources comme des variables continues. Afin de guider notre agent dans la réalisation de ses objectifs, nous proposons d'effectuer une **délibération locale** et non plus complète, et la mise en place d'un cycle délibération / exécution. Un cycle débutera en calculant à partir de la situation actuelle une politique locale, appelée

agenda des macro-actions, qui donne pour un horizon donnée les macro-actions à exécuter. L'agent exécutera ensuite ce plan jusqu'à son terme puis commencera un nouveau cycle. Nous présenterons les différentes méthodes que nous utilisons afin de :

1. **Calculer les transitions entre deux états globaux** : les transitions entre deux états globaux doivent désormais prendre en compte les *rmt* dont la condition est établie sur les niveaux de ressources. Pour cela, nous utilisons les calculs des coûts moyens en ressources des π^{msv}_{rmt} pour estimer la probabilité que le seuil en ressource d'une *rmt* soit dépassé.
2. **Construire un arbre local des macro-actions** : nous construisons un arbre où chaque nœud est un état global, la racine étant l'état global au début du cycle. Nous développons un nœud en calculant les états futurs pour toutes les π^{msv}_{rmt} qu'il est possible d'exécuter depuis ce nœud. L'arbre est développé jusqu'à un horizon exprimé à l'aide de la ressource de temps.
3. **Calculer un agenda de macro-actions** : nous élaguons l'arbre précédemment calculé afin de sélectionner pour chaque nœud la macro-action promettant la récompense la plus importante.
4. **Exécuter un agenda de macro-actions** : l'agenda de macro-actions calculé est ensuite exécuté. Pour cela, nous exécutons les macro-actions correspondantes aux nœuds atteints. L'exécution s'arrête lorsqu'une feuille est atteinte.

Ces différentes étapes seront détaillées dans ce chapitre.

5.1.2 Définition des ressources

Afin de rendre un agent autonome dans le choix de ses objectifs, nous avons proposé un système modélisant de façon originale les objectifs de l'agent, à l'aide de motivations. Ces objectifs intègrent les notions de *permanence* (la motivation *Maintien du niveau de la batterie* par exemple), et d'*enchaînement* de tâches. Nous introduisons deux ressources qui sont souvent considérées dans le contexte de la robotique autonome :

- **L'énergie** : correspond à la consommation énergétique. Cette ressource est importante dans un contexte de missions longues, ou lorsque le choix des actions peut être guidé par leur coût énergétique.
- **Le temps** : la multiplication des objectifs associée à une durée d'expérience limitée pose le problème de la temporalité. Il s'agit en premier lieu de remplir au mieux les objectifs sur une période donnée, en exécutant des actions dont la durée peut varier. Le temps de résolution d'une tâche devient alors un facteur déterminant. De plus, certains objectifs peuvent également être directement dépendants du temps. Cela peut être le cas pour exprimer un délai pendant lequel une tâche est réalisable, ou afin de modéliser des tâches récurrentes.

L'introduction de ces deux ressources nous pose le problème de la considération des variables continues. L'énergie et le temps se représentent de façon naturelle à l'aide de variables numériques continues, dont la valeur est incrémentée ou décrémentée en fonction des actions exécutées par l'agent. Dans la littérature, les MDPs qui incorporent de telles variables sont dits *hybrides* [Meuleau et al., 2009] et l'état de ces systèmes se note

$s = (x, y)$, x étant la composante discrète de l'état, identique aux états habituels, et y étant la composante continue, où est indiquée la valeur numérique de chaque état continu. Les MDPs hybrides sont résolus en discrétisant les variables continues à l'aide de plusieurs techniques. La plus simple consiste en la discrétisation de chacune des variables en définissant à priori des intervalles de valeur. Des techniques plus complexes consistent à découper l'espace multi-dimensionnel des variables continues en hypercubes ou hyper-rectangles, permettant un découpage plus fin et plus efficace [Feng et al., 2004]. Une fois le nouvel espace d'état formé, la fonction de transition est constitué en évaluant la probabilité pour les ressources de passer d'un intervalle de valeur à un autre, et donc un autre état. Cependant, ces techniques permettent de retourner à un problème MDP classique au prix d'une augmentation importante de l'espace d'état. De plus, un type de variable continue vient compliquer la situation : les **variables continues non bornées**, comme le temps. En effet, ces variables, même discrétisées, impliquent la création d'une infinité d'états. Dans ce cas, les méthodes complètes (type Value Iteration) sont rendues inutilisables, en raison de la nécessité d'expliciter la totalité des états. Les méthodes locales sont alors les seules utilisables, comme par exemple l'algorithme heuristique Hybrid AO* [Mausam et al., 2005].

Il existe des cas où les variables continues ne sont considérées que comme des paramètres à optimiser [Kolobov, 2012]. Dans ce cas, il est possible de ne pas inclure les variables continues dans l'espace d'état, mais de les prendre en compte en les incorporant dans les algorithmes de recherche. Si nous ne cherchons qu'à exprimer le temps, il est ainsi possible de rechercher une solution complète à un MDP qui maximisera non pas la récompense seule, mais la récompense par unité de temps passé [Bradtke and Duff, 1995]. Cependant, notre problème exige de prendre en compte l'énergie et le temps comme variables à minima dans les motivations. Le niveau de la batterie sera surveillé pour éviter qu'il tombe à zéro et déterminera s'il est judicieux ou non de se recharger. Le temps sera quant à lui utilisé afin d'exprimer des dates butoirs ou des tâches récurrentes. Ces deux variables ne peuvent donc pas être entièrement réduites à des variables à optimiser.

Pour répondre à ce problème, nous avons choisi de définir nous-même la notion de ressource, afin de pouvoir l'intégrer au sein des motivations et de trouver une solution au problème posé. Une ressource r sera traitée comme suit :

- **conservation d'une variable continue** : le problème des ressources n'étant pas adressé au niveau du calcul des politiques résolvant les *rmt*, nous conservons des variables continues pour simplifier les calculs de la partie délibération.

- **ressources comme condition des transitions d'une motivation** : le niveau des ressources peut, au même titre que les variables d'état, conditionner une *rewarded motivation transition rmt*. Dans ce cas, la condition de *rmt* spécifie une expression mathématique faisant intervenir une ou plusieurs ressources. Les ressources sont par exemple utilisées pour définir des objectifs comme conserver un niveau d'énergie satisfaisant ou accomplir des tâches contraintes par le temps. La *rmt* se déclenche lorsque la-dite expression devient vraie.

- **non influence des ressources dans la fonction de transition** : nous choisissons de considérer que le niveau des ressources n'a pas d'influence dans la fonction de transition du modèle du monde et dans le calcul des politiques π_{rmt}^{msv} au sein du module opérationnel. En d'autres termes, cela signifie que pour déterminer les effets d'une action sur l'état du monde, le niveau des ressources n'interviendra pas (voir

Section 4.3.1). L'exécution d'une action aura en revanche un effet sur le niveau des ressources.

Nous pensons en effet que l'influence du niveau des ressources a un effet négligeable sur la réalisation des actions primitives. Cette influence sera prise en compte dans un second temps, dans le processus de délibération, lors du choix de la meilleure macro-action à exécuter. Remarquons que le choix de ne pas considérer l'influence des ressources sur l'exécution des actions nous oblige à ignorer le cas où la batterie est vide, pour lequel naturellement aucune action n'est réalisable.

- **fonction de coût** : le coût en ressource d'une action est donné par la fonction de coût $C_r(ws, a, ws')$. Le coût en ressource d'une action dépend donc bien sûr de la nature de l'action, mais peut également dépendre de l'état du monde ws dans lequel est exécuté cette action ainsi que ws', l'état dans lequel le monde se trouve après son exécution.

  La fonction $C(ws, a, ws')$ donne le coût en ressource pour la transition $[ws, a, ws']$ et pour toutes les ressources r.

- **non recherche de politique pour les** rmt **conditionnées par des ressources** : dans le contexte où nous calculons des politiques complètes pour déclencher des rmt, nous ne disposons pas de moyens pour calculer une politique dont l'objectif est de porter une ressource à un niveau donné.

  Si les techniques de planification locale permettent de calculer des plans visant un niveau de ressource, nous ne connaissons pas de méthodes complètes le permettant. À notre connaissance, la seule possibilité pour prendre en compte les ressources dans les méthodes complètes est de les utiliser comme facteur d'optimisation (voir par exemple [Svorenova et al., 2013]). Pour donner un exemple, il est possible de rechercher des politiques maximisant la récompense obtenue par unité de temps ou minimisant la dépense de batterie. Il n'est cependant pas possible à notre connaissance de chercher une politique dont l'objectif serait de passer le niveau de la batterie au-dessus d'une valeur donnée.

  En conséquence, les motivations ayant pour but de changer un niveau de ressource doivent utiliser des moyens détournés. Dans ce cas, nous définissons une rmt supplémentaire, généralement bouclante, dont le rôle est de valoriser les comportements permettant de se rapprocher de l'objectif. Dans la motivation de la **Survie**, tirée de notre exemple récurrent, présenté Figure 3.8, le passage de l'état motivationnel *faible* à *bon* illustre bien ce raisonnement. La condition de la *rewarded motivation transition* entre *faible* et *bon* est définie comme "passer au-dessus du niveau de batterie 50%". Puisqu'aucune politique ne peut être calculée pour cette tâche, une seconde rmt est mise en place depuis *faible* et consiste à recharger la batterie une fois. De cette manière, en valorisant le fait de recharger la batterie, la rmt de *faible* à *bon* pourra être déclenchée pendant l'exécution de notre seconde rmt.

Pour finir, nous définissons les deux ressources que nous utilisons dans notre exemple récurrent, puis dans les résultats expérimentaux :

- **energy** : correspond au niveau de la batterie, dont la valeur numérique est comprise dans l'intervalle $[0 - 100]$, correspondant au pourcentage de batterie restant. Si le niveau de la batterie arrive à 0, alors l'expérience prend fin, l'agent ayant vidé sa batterie.

- **time** : correspond au temps passé depuis le début de l'expérimentation. Le temps s'exprime en unité de temps, abrégé **ut** (pour *unit of time*). Il démarre à 0 lors de l'initialisation de l'expérimentation, ou tout autre date voulue.

5.1.3 Architecture

Suite à la problématique des ressources et donc l'introduction de variables continues dans notre problème, l'architecture présentée Section 4.1.2 doit évoluer pour prendre en compte les changements apportés.

En effet, les variables continues représentant les niveaux des ressources créent une infinité d'état globaux possible. Pour calculer une solution, nous utilisons donc une méthode de recherche locale afin d'obtenir un **agenda de macro-actions**, c'est-à-dire un plan pour une durée limitée. Puisqu'il est limité dans le temps, un nouvel agenda doit être produit périodiquement afin de résoudre un problème dont la durée, potentiellement infinie, dépasse celle d'un agenda. En conséquence, pour prendre cet aspect en compte, nous choisissons de représenter dans l'architecture les éléments présentant l'exécution de la solution.

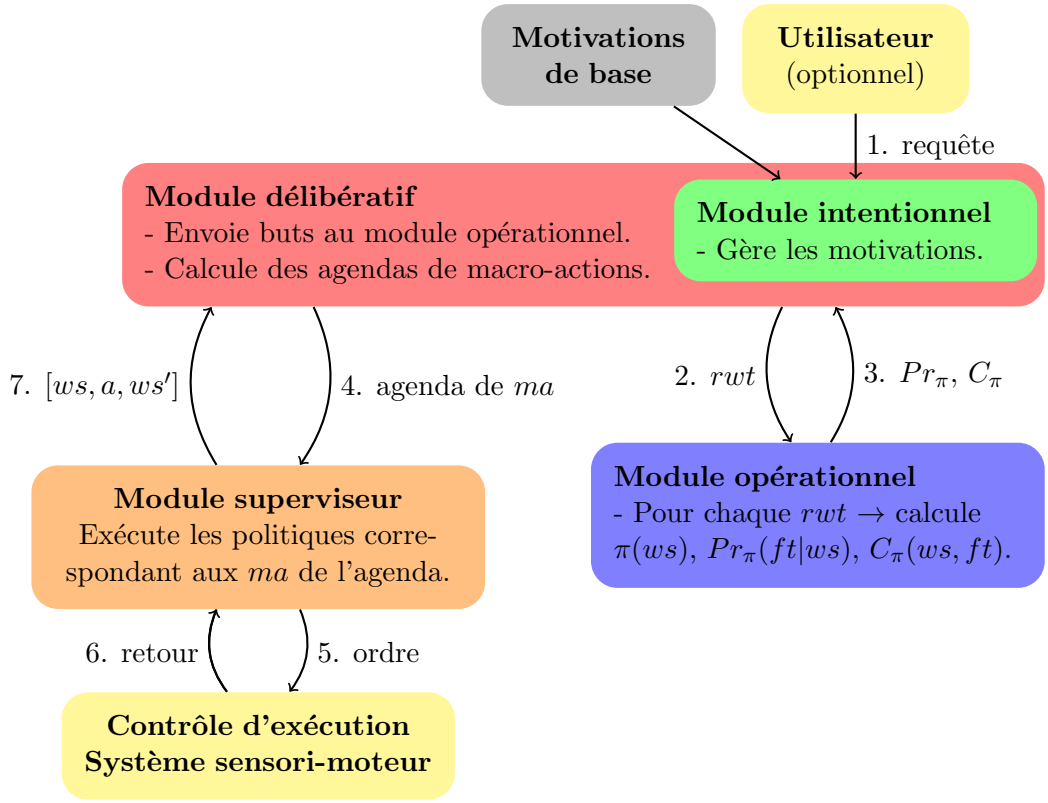


Figure 5.1: Présentation de l'architecture (version avec ressources), composée des modules opérationnel, intentionnel et délibératif.

Pour représenter la résolution du problème, nous introduisons un cycle délibération / exécution au travers du module superviseur et du système sensori-moteur, dans la Figure 5.1. Le module superviseur assure l'exécution des agendas de macro-actions que lui apporte le module délibératif et lui renvoie l'évolution du monde après l'exécution

de chaque action. Pour exécuter un agenda de macro-actions, le module superviseur demande la politique correspondant à la macro-action ma au module opérationnel.

5.2 Déterminer le coût moyen en ressource d'une macro-action

Afin de modéliser l'exécution d'une politique π^{msv}_{rmt} sous la forme d'une macro-action, nous avons pour l'instant calculé $Pr(rwt|ws, \pi)$ (la probabilité de finir par l'exécution de rwt, pour chaque rwt) et $NbAction(rwt|ws, \pi)$ (le nombre d'actions moyen utilisé pour finir par rwt, pour chaque rwt). Si le nombre d'actions moyen pour exécuter une politique ne nous sera plus utile dans le contexte de ce chapitre, nous souhaitons connaître le coût en ressource accumulé lors de son exécution.

5.2.1 Calcul de la fonction $C_r(rwt, ws, \pi)$

Le calcul du coût moyen d'une politique pour la ressource r est très proche du calcul de $NbAction(rwt|ws, \pi)$ (voir Section 4.3.6 pour plus de détails) et nécessite également le calcul préalable de $Pr(rwt|ws', \pi)$. La Figure 5.2 donne l'algorithme permettant de construire le système d'équations pour le calcul de $C_r(rwt, ws, \pi)$. La principale et seule différence avec le calcul précédent est l'initialisation de la valeur de $C_r(rwt, ws, \pi)$ dans le système d'équation. Celle-ci prend maintenant en compte de $C_r(ws, \pi(ws), ws')$, le coût en ressource pour la transition $[ws, \pi(ws), ws]$ pour r.

Exemple

Pour développer notre exemple, nous poursuivons l'exemple débuté Section 4.3.8, en calculant le coût moyen en ressource de la politique afin de constituer une macro-action utilisable. La figure 5.3 rappelle la chaîne de Markov correspondant à la politique $\pi_{recharge}$ et indique sur chaque transitions le coût de la ressource temps $C_{time}(ws, \pi(ws), ws')$ en plus de la probabilité de sa réalisation. Le Tableau 5.1 donne quant à lui le système d'équations pour calculer $C_{time}([Eg, recharge, Eu], ws, \pi_{recharge})$ ainsi que les résultats de $C_{time}(rwt, ws, \pi_{recharge})$ pour les cinq rwt possibles. Les équations sont écrites dans des tableaux de la manière suivante :

	A	B	D	E	F	H	S	$C_{time}(rwt, ws, \pi)$
B	0.05	0.20	0	0	0.05	0	0	0.70

correspond à (avec $f(ws) = C_{time}(rwt, ws, \pi)$) :
$f(B) = 0.05 \times f(A) + 0.20 \times f(B) + 0.05 \times f(F) + 0.70$

5.2.2 Générer une distribution des coûts en ressources

Cependant, le coût moyen en ressource d'une politique π et pour un état ws n'est pas suffisant pour déterminer les effets d'une macro-action. En effet, prenons le cas d'une d'un état de départ $gs = (ws, msv)$ et avec $time = t$. Nous souhaitons déterminer les effets d'une macro-action ma, et de savoir si une rmt dont la condition est $(time > t + x$ sera déclenchée ou non lorsque la *rewarded world transition* rwt est atteinte. Or, comme l'illustre la Figure 5.4, le coût en ressource d'une politique est aléatoire. Dès lors, si le coût

1: $lignes = [ws \in WS_{red}]$
2: $colonnes = [ws \in WS_{red}]$
3: $tableau = [lignes][colonnes]$
4: ▷ initialise un tableau représentant un système d'équations
5: **for** $ws \in lignes$ **do**
6: **for** $ws' \in colonnes$ **do**
7: $tableau[ws][ws'] = T(ws, \pi(ws), ws')$
8: ▷ initialise le tableau avec la chaîne de Markov
9: **end for**
10: **end for**
11: **for** $ws \in lignes$ **do**
12: **for** $ws' \in colonnes$ **do**
13: $tableau[ws][ws'] = tableau[ws][ws'] \times Pr(rwt|ws', \pi)$
14: ▷ pondère la transition par la probabilité, depuis ws' de finir par rwt
15: **if** $(ws, \pi(ws), ws') \in RWT$ **then**
16: $tableau[ws][ws'] = 0$ ▷ supprime les transitions finales du tableau
17: **end if**
18: **end for**
19: **end for**
20: $tableau[ws_{rwt}][ws'_{rwt}] = T(ws_{rwt}, \pi(ws_{rwt}), ws'_{rwt})$
21: ▷ rétablit uniquement la transition de rwt, non pondérée
22: *rajouter colonne* $C_r(rwt, ws, \pi)$
23: **for** $ws \in lignes$ **do**
24: $total = 0.00$ ▷ compteur des valeurs du tableau
25: **for** $ws' \in colonnes$ **do**
26: $total = total + tableau[ws][ws']$
27: **end for**
28: **if** $total \neq 0$ **then**
29: **for** $ws' \in colonnes$ **do**
30: $tableau[ws][ws'] = tableau[ws][ws']$ / $total$
31: ▷ rétablit à 1 la somme des probabilités des transitions de ws
32: $tableau[ws][C_r(rwt, ws, \pi)] \mathrel{+}= C_r(ws, \pi(ws), ws') \times tableau[ws][ws']$
33: ▷ initialise le coût moyen en partant de ws
34: **end for**
35: **end if**
36: **end for**
37: $tableau[ws_{rwt}][ws'_{rwt}] = 0$
38: ▷ supprime la transition de rwt

Figure 5.2: Algorithme *ConstruireSysCMoy*, constituant le système d'équations de $C_r(rwt, ws, \pi)$.

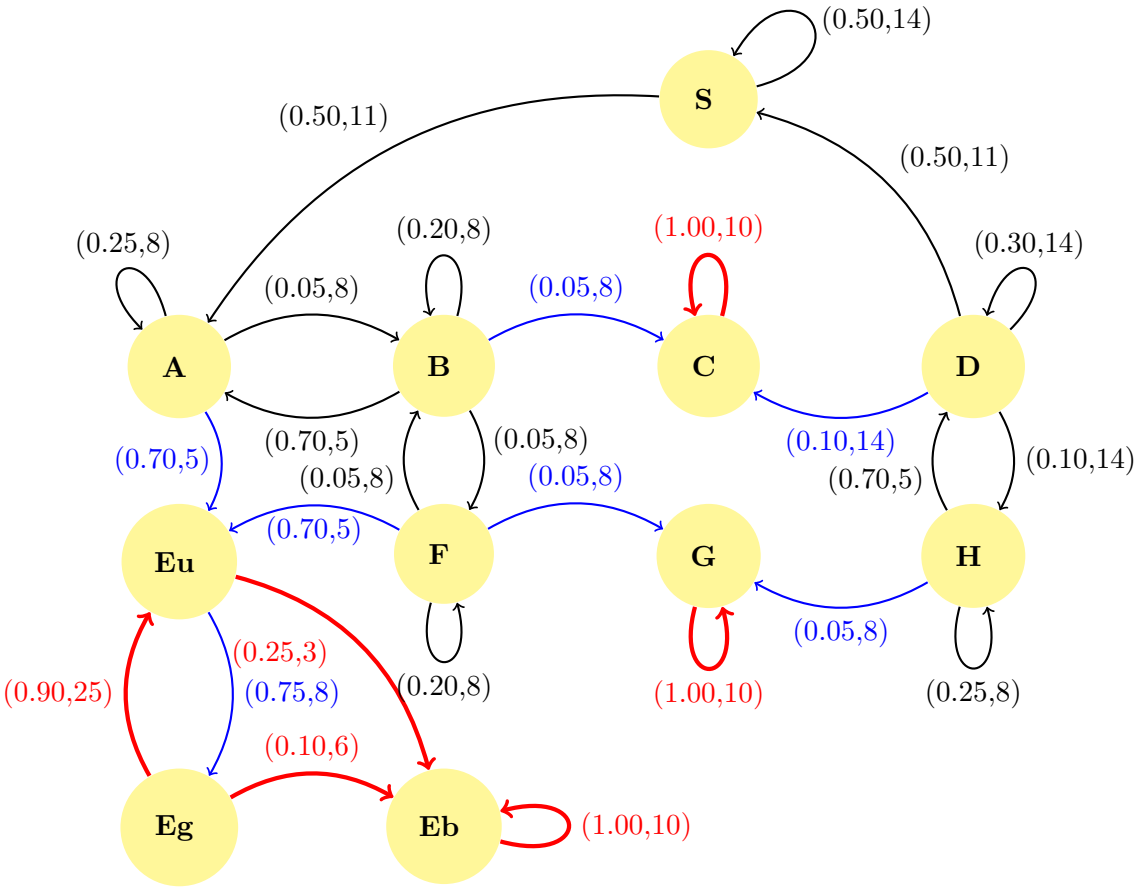


Figure 5.3: Représentation de la politique $\pi_{recharge}$ comme une chaîne de Markov. Les transitions sont annotées d'un couple $(pr(ws, \pi(ws), ws'), C_{time}(ws, \pi(ws), ws'))$.

moyen peut être calculé, le coût lors d'un essai réel est variable et en fonction de ce coût, différents scénarios peuvent se présenter. En conséquence, ce n'est pas du coût moyen en ressource dont nous avons besoin, mais d'une distribution des coûts.

Nous nous intéressons à la distribution des scénarios et de leurs coûts respectifs. Les scénarios n'ayant pas d'ordre naturel, nous décidons d'organiser les distributions en triant les scénarios selon leur coûts en ressource, que nous appelons à présent **distribution ascendante des coûts**.

Distribution ascendante des coûts pour (π, rwt). Notre problème est à présent de déterminer cette distribution des coûts afin de pouvoir connaître la probabilité que l'exécution de ma ait un coût en ressource de temps inférieur ou supérieur à x. Nous allons pour cela nous intéresser à la distribution ascendante des coûts d'une distribution depuis un état ws pour une politique π et rwt. Illustrée Figure 5.5 pour $\pi_{recharge}$ et depuis l'état H, cette courbe donne, pour un coût en ressource c, la probabilité que l'exécution de la politique ait un coût inférieur à c lorsqu'elle termine par rwt (ici $(Eg, recharge, Eu)$).

Pour calculer cette distribution, une solution consiste à déterminer l'ensemble des différents chemins que la politique est susceptible d'emprunter en ensuite d'ordonner ces scénarios en fonctions de leurs coûts respectifs. Cependant, si la chaîne de Markov représentant l'exécution de la politique contient une boucle, alors le nombre de scénarios

Équations initiales pour $rwt = (Eg, recharge, Eu)$:

ws	A	B	C	D	Eu	Eg	Eb	F	G	H	S	$C_{time}(rwt, ws, \pi)$
A	0.25	0.05	0	0	0.7	0	0	0	0	0	0	5.89
B	0.75	0.2	0	0	0	0	0	0.05	0	0	0	5.75
C	0	0	0	0	0	0	0	0	0	0	0	0.00
D	0	0	0	0	0	0	0	0	0	0.09	0.61	12.18
Eu	0	0	0	0	0	1.00	0	0	0	0	0	3.00
Eg	0	0	0	0	0	0	0	0	0	0	0	25.00
Eb	0	0	0	0	0	0	0	0	0	0	0	0.00
F	0	0.05	0	0	0.75	0	0	0.2	0	0	0	5.75
G	0	0	0	0	0	0	0	0	0	0	0	0.00
H	0	0	0	0.75	0	0	0	0	0	0.25	0	5.75
S	0	0.5	0	0	0	0	0	0	0	0	0.5	12.5

$rwt1 : (Eu, verify, Eb)$, $rwt2 : (Eg, recharge, Eu)$,
$rwt3 : (Eg, recharge, Eb)$, $rwt4 : (C, wait, C)$, $rwt5 : (G, wait, G)$

ws	$C_{time}(rwt, ws, \pi)$				
	$rwt1$	$rwt2$	$rwt3$	$rwt4$	$rwt5$
A	16.85	36.85	17.85	31.85	41.85
B	24.0	44.0	25.0	21.18	31.18
C				10.0	
D	70.25	90.25	71.25	42.55	52.66
Eu	8.0	28.0	9.0		
Eg		25.0	6.0		
Eb					
F	16.18	36.18	17.18	31.18	20.09
G					10.0
H	77.92	97.92	78.92	50.22	27.33
S	49.0	69.0	50.0	46.18	56.18

Table 5.1: Résultats du calcul de $C_r(rwt, ws, \pi)$ pour $\pi_{recharge}$ et la ressource *time*.

possibles devient infini, tout comme le coût en ressource maximal. La Figure 5.6 illustre ce phénomène. Dans le cas de gauche, qui ne contient pas de boucle, il y a quatre scénarios facilement identifiables : $[A \to C \to D \to E]$, $[A \to B \to C \to D \to E]$, $[A \to B \to D \to E]$ et $[A \to D \to E]$, dont les coûts respectifs sont 5, 15, 29 et 35. Dans le cas de droite, il n'est plus possible de parcourir tous les scénarios possibles, ce que montre la distribution des coûts constatés. En effet, Nous pouvons voir une multitude de

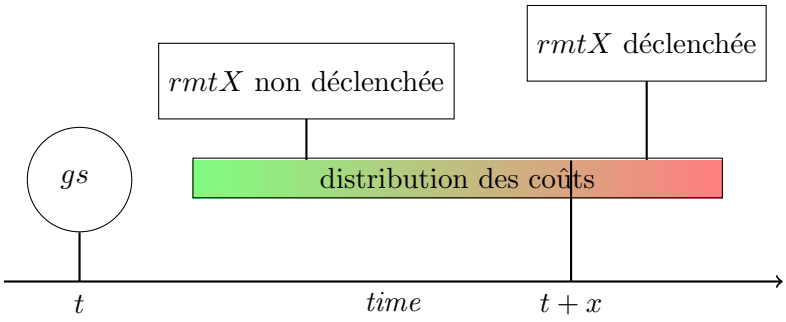


Figure 5.4: Schéma de l'effet d'une macro-action sur le déclenchement d'une *rmt* se déclenchant lorsqu'une niveau de ressource est atteint.
Dans la situation présentée, $rmtX$ se déclenche à $time = t+x$. Selon le coût de la politique, aléatoire à cause du modèle de transition stochastique, $rmtX$ peut être déclenchée ou non.

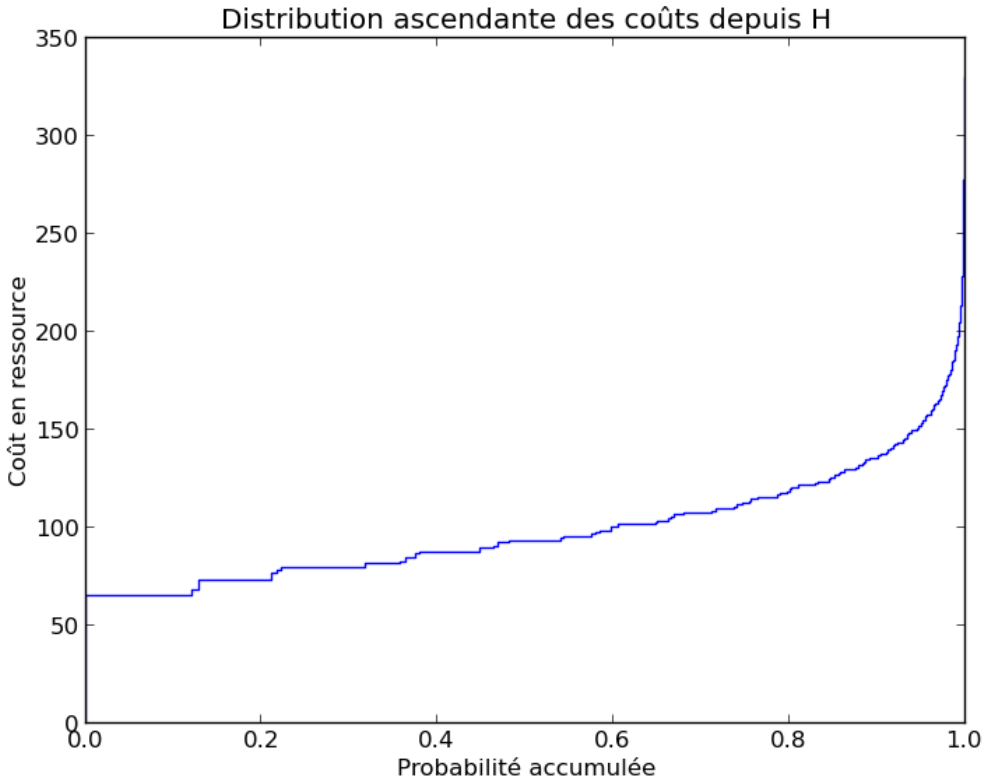


Figure 5.5: Distribution ascendante des coûts depuis H pour la politique $\pi_{recharge}$ et pour $(Eg, recharge, Eu)$.
Cette courbe a été obtenue en effectuant 500 000 tests de la politique $\pi_{recharge}$ dont le coût des tests finissant par la rwt $(Eg, recharge, Eu)$ ont été compilés.

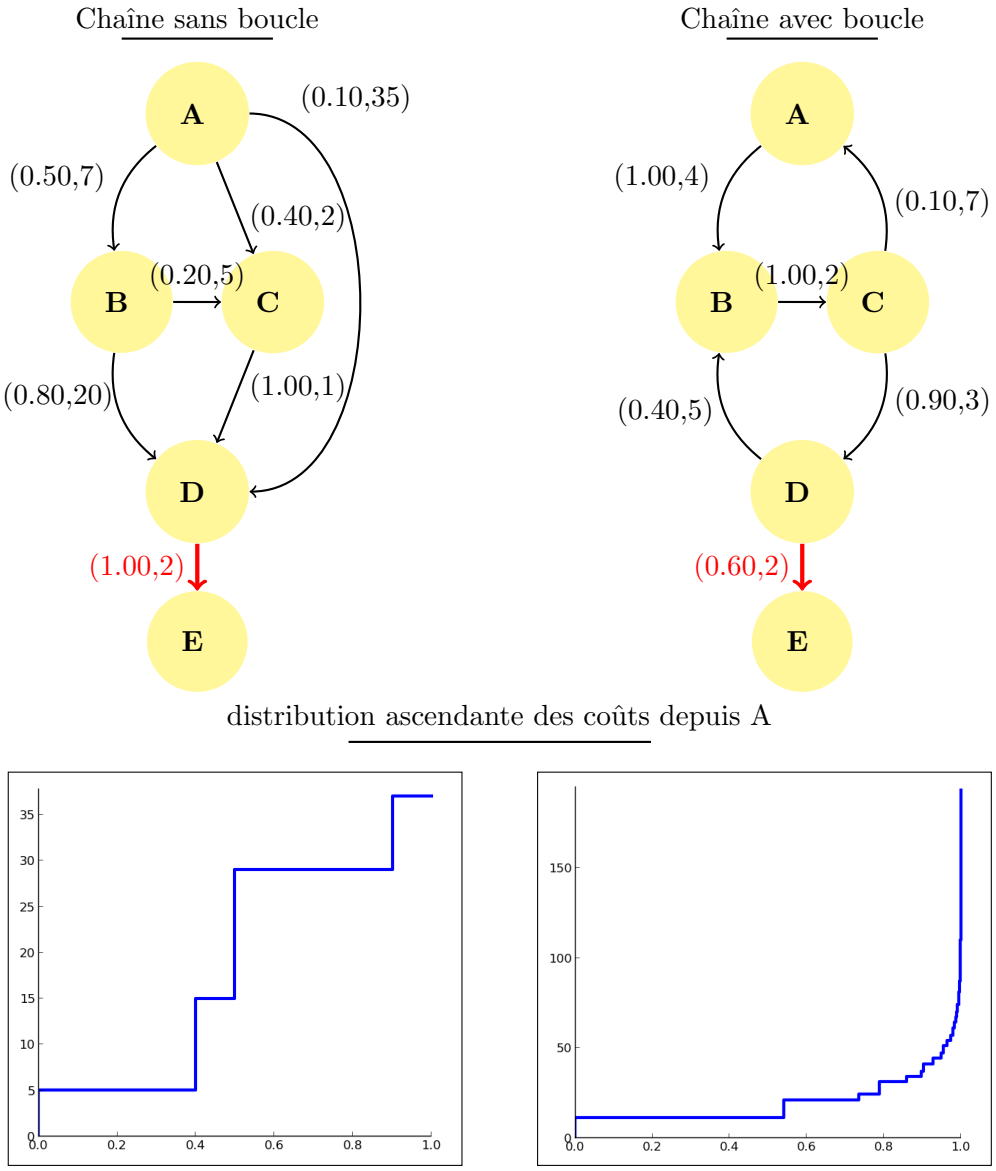


Figure 5.6: Schéma de la distribution ascendante des coûts avec ou sans boucle. Les distributions ascendantes des coûts ont été effectuées sur la base de 500 000 tests chacun, partant de A et en finissant dans E.

scénarios peu probables et dont le coût est de plus en plus important à la fin de la courbe.

Donner une approximation de la distribution ascendante des coûts pour (π, rwt).

Déterminer cette la distribution ascendante des coûts pour (π, rwt) pour une politique π ne présentant aucune boucle est envisageable. Cependant, en présence de boucles, il n'est pas possible de dénombrer suffisamment de scénarios possibles pour obtenir une distribution convaincante en un temps raisonnable, or les boucles sont extrêmement courantes dans les politiques résolvant les problèmes auxquels nous nous adressons. Nous cherchons donc une méthode permettant, dans un temps raisonnable, de produire une approximation de

cette distribution.

Pour produire une approximation de cette distribution, nous avons besoin d'un nombre de points minimum. Compte-tenu du profil des courbes dès lors qu'il y a des boucles dans la politique, il nous semble important de donner un point à $pr = 0$, puis plusieurs points de plus en plus rapprochés de $pr = 1$. Nous cherchons à obtenir une méthode permettant d'obtenir un point de la courbe. En l'absence d'une méthode exacte et rapide, nous cherchons à en obtenir une approximation.

Obtenir un point approximatif sur la distribution.

Pour commencer, nous pouvons calculer le point placé à $pr = 0$ de façon simple et exacte. En effet, le premier point de la courbe correspond au scénario le moins coûteux en ressources. Or, cette valeur peut être calculée à l'aide de l'algorithme de Dijkstra. En ne gardant sur le graphe que les valeurs de coûts, le plus court chemin, l'algorithme donnera pour chaque état la valeur en ressource du chemin le moins coûteux. Nous exécutons cet algorithme en partant de chaque arrivée (les *rewarded world transitions*) et en effectuant la mise à jour des plus courts chemins "à l'envers". Cela nous permet ainsi de calculer le plus court chemin pour tous les états possibles en une seule fois.

Pour obtenir les points suivants, nous explorons trois méthodes permettant de les déterminer. Dans les trois cas, nous considérons qu'obtenir une précision acceptable depuis $pr = 0$ jusqu'à $pr = 0.99$ est suffisant. La dernière partie de la courbe donnant un ensemble de cas très peu probables, ceux-ci pourront être ignorés par la suite, dans les calculs de l'agenda de macro-actions (voir 5.3.2).

Énumération. La première méthode correspond à ce qu'il est possible de faire aisément pour la chaîne sans boucle de la Figure 5.6, c'est-à-dire énumérer les scénarios. Le but est ici d'en énumérer suffisamment pour couvrir un pourcentage donné des différents scénarios. Ainsi, en présence de boucle, l'énumération ne pouvant finir, nous pouvons choisir de nous arrêter lorsque la somme des probabilités des scénarios déjà énumérés dépasse la valeur choisie. Les données sont ensuite triées et nous obtenons la courbe entière, tronquée des scénarios les moins probables.

Si cette méthode s'avère exacte, obtenir une bonne approximation pour les cas s'approchant de $pr = 1$ nécessite une quantité importante de calculs. De plus, le calcul s'effectue depuis un unique état de départ et pour une unique rwt. Cette méthode se révèle donc très coûteuse et sa fiabilité devient faible pour les points proches de $pr = 1$.

Distribution test. La seconde méthode consiste à utiliser des distributions obtenues à l'aide de tests statistiques. Ce sont ces tests statistiques que nous avons utilisé précédemment pour illustrer les distributions ascendantes de nos exemples.

Si ces distributions tests demandent un grand nombre de tests pour être considérées comme fiable, un nombre de tests plus réduits permettrait d'avoir une estimation suffisante. Cependant, comment connaître le nombre de tests suffisant pour avoir une distribution fiables? De plus, comme pour la méthode de l'énumération, les distributions test doivent être calculées pour chaque état possible ws.

Pondération de la fonction de transition. Pour la troisième méthode, nous utilisons une calcul est inspiré de celui de $C_r(rwt, ws, \pi)$. Une étape importante du calcul

de $C_r(rwt, ws, \pi)$ consiste à pondérer la valeur des transitions $[ws, a, ws']$ par la probabilité pour ws' de finir par rwt, c'est-à-dire $Pr(rwt|ws', \pi)$ (voir ligne 13 de l'algorithme *ConstruireSysCMoy* Figure 5.2). Si cette pondération permet d'obtenir la fonction de transition effective lorsque la politique termine par rwt, nous essayons d'effectuer une seconde pondération afin d'obtenir des points de la distribution ascendante des coûts. En effet, lorsque nous pondérons à 1 les transitions qui correspondent au chemin de coût minimum en ressource et les autres transitions à 0, alors la résolution du système d'équations donne le même résultat que l'algorithme de Dijkstra. De plus, si nous pondérons à 1 les transitions afin de conserver les boucles (représentant donc le pire cas, où l'état parcoure la boucle sans fin) alors les états pour lesquels il est possible d'entrer dans les boucles ne convergent plus. Pour ces états, le système d'équations exprime donc que le coût pour cet état, dans cette configuration, est infini.

Nous avons donc cherché une méthode permettant de pondérer la fonction de transition et qui permette d'obtenir un point de la distribution ascendante des coûts. Cependant, le calcul se révèle inexact et l'écart à la véritable courbe peut varier grandement dans certains cas. Néanmoins, ce calcul est facile à réaliser, rapide, et donne la plupart du temps une valeur dont l'ordre de grandeur est bonne. La Figure 5.7 donne le cas de la distribution ascendante des coûts pour la politique $\pi_{recharge}$ de notre exemple récurrent, pour atteindre la rwt $[Eg, recharge, Eu]$ et depuis l'état H. L'approximation renvoyée pour cet état et la quasi-totalité des états de cet exemple sont suffisamment corrects pour être exploités. En l'absence d'une alternative utilisable, nous utilisons cette méthode arbitraire afin d'obtenir une approximation de la distribution ascendante des coûts.

Distributions liées des coûts pour des ressources multiples $PrInfC_r(c|rwt, ws, \pi)$

Si nous sommes à présent capable d'estimer la probabilité que le coût d'une ressource soit inférieur à un coût c donné, nous devons aborder le problème de la combinaison de plusieurs ressources.

En effet, prenons un cas où nous avons deux rmt pouvant être déclenchées par l'exécution de $\pi_{recharge}$:

- **rtm1** : dont la condition dépend de la ressource de temps et se déclenche dans 80**ut**.
- **rmt2** : dont la condition dépend de la ressource d'énergie et se déclenche après l'utilisation de 2% de la batterie.

Dans cette situation, nous pouvons nous aider des estimations des distributions de coûts ascendants pour connaître la probabilité, pour chaque rmt d'être déclenchée. À partir des distributions de ces deux ressources (Figure 5.8), nous obtenons :

- **rtm1** : une probabilité estimée de 0.60 d'être déclenchée (probabilité constatée : 0.68).
- **rmt2** : une probabilité estimée de 0.14 d'être déclenchée (probabilité constatée : 0.29).

Cependant, ce n'est pas la probabilité de déclenchement des rmt que nous souhaitons, mais la probabilité de chaque combinaison de déclenchement possible :

rmt1/rmt2	non déclenchée	déclenchée
non déclenchée	?	?
déclenchée	?	?

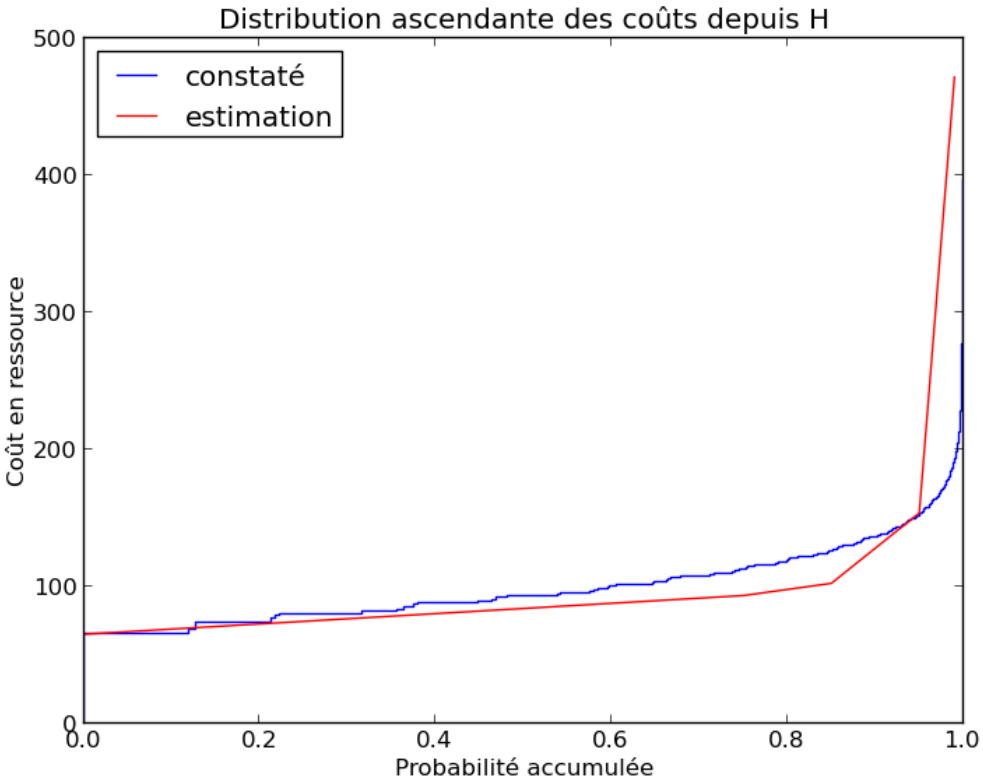


Figure 5.7: Distribution ascendante des coûts pour $\pi_{recharge}$, depuis l'état H et terminant par $[Eg, recharge, Eu]$, comparaison entre distribution constatée et estimation.
La courbe bleue donne la distribution constatée construite à l'aide de 500 000 tests, échantillon considéré comme suffisant pour obtenir une distribution fiable. La courbe rouge est constituée à partir de cinq points, disposés à $pr = 0, 0.75, 0.85, 0.95, 0.99$. Le point à $pr = 0$ a été calculé à l'aide du plus court chemin, les autres à l'aide de la méthode de la pondération de la fonction de transition.

Pour répondre à cette question, il nous faut trouver un moyen pour corréler les deux estimations entre elles, c'est-à-dire de lier les consommations pour les deux ressources. Cela revient à se pose la question : "Si la politique termine en plus de 80 **ut**, quel est la probabilité que le la consommation de batterie soit supérieure à 2%?".

Pour corréler les coûts des deux ressources, nous avons choisi de ne plus nous intéresser à une ou des ressources séparément, mais à une *fonction de ressource*. Une fonction de ressource est une fonction donnant une somme pondérée des coûts pour les ressources du problème. Celle que nous avons choisi pour cet exemple est :

$$cf = 1 \times time + 8 \times energy$$

Ces coefficients permettent de mieux prendre en considération la ressource d'énergie, dont les coûts ont une valeur plus faible. Nous utilisons ensuite cette fonction à la place d'une ressource unique pour calculer les chemins de coût minimum et maximum à la place des ressources, et enfin calculer une nouvelle distribution des coûts. Celle-ci fait figurer

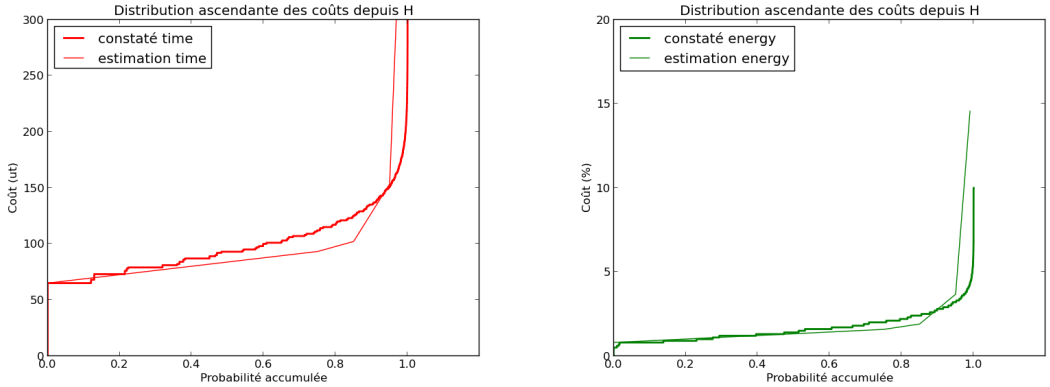


Figure 5.8: Distributions ascendante des coûts en ressource de temps et énergie. La figure de gauche donne les coûts ascendants constatés en temps (trait épais) et notre estimation (trait fin), pour $\pi_{recharge}$, depuis l'état H et terminant par $rwt = [Eg, recharge, Eu]$. La figure de droite donne la même donnée pour la ressource d'énergie (les coûts pour la ressource d'énergie ont été définis arbitrairement).

la distribution ascendante de la fonction de ressource et associe en chaque point de la courbe la consommation des deux ressources correspondant à la valeur de la fonction de ressource. En procédant de cette manière, nous lions les deux ressources à la fonction de ressource. La Figure 5.9 montre cette triple distribution, la valeur de la courbe ascendante de la fonction de ressource est la somme pondérée des valeurs des courbes correspondant aux coûts en énergie et en ressource de temps. Comme pour les distributions précédentes, il est possible de calculer des estimations pour les différentes ressources de la distribution. Nous utilisons pour cela les techniques utilisées précédemment.

Nous souhaitons à présent exploiter l'estimation de cette distribution liée pour déterminer la probabilité des différentes combinaisons de déclenchement de rmt. Nous pouvons voir que les coûts des ressources *time* et *energy* ne sont plus monotones. Cependant, nous pouvons constater que les courbes des ressources sont hachées, mais suivent des dynamiques que nous pouvons exploiter. Pour déterminer la probabilité de chaque combinaison de déclenchement d'un ensemble de rmt, nous déterminons sur quelles plages les rmt sont ou non déclenchées.

Test. Revenons à notre exemple, avec $rmt1$ ($time >= 80$**ut**) et $rmt2$ ($energy >= 2\%$) pour l'exécution de $\pi_{recharge}$ terminant par $rwt = [Eg, recharge, Eu]$. Lors de la génération de la distribution test, nous avons constaté les données suivantes:

rmt1/rmt2	non déclenchée	déclenchée
non déclenchée	0.32	0.0
déclenchée	0.39	0.29

À présent, nous allons utiliser la distribution liée des ressources pour estimer la probabilité des combinaisons de déclenchements de $rmt1$ et $rmt2$. La Figure 5.10 schématise la manière dont nous segmentons la distribution pour déterminer les combinaisons possibles et leur probabilité. Notre estimation donne donc:

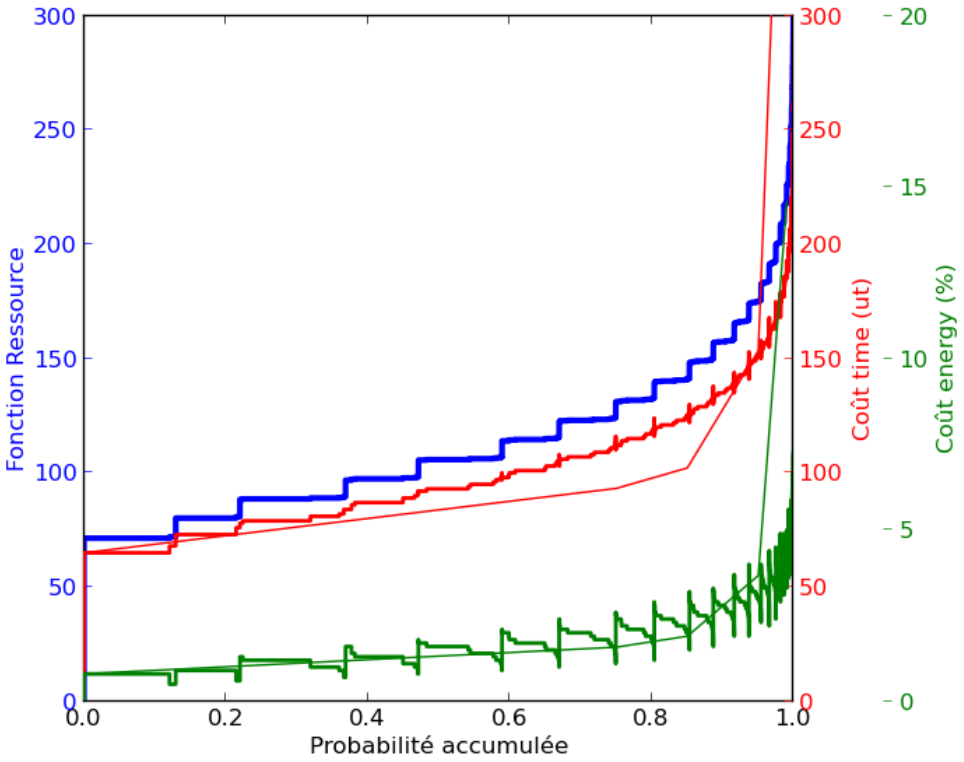


Figure 5.9: Distributions liées des coûts en ressources.

rmt1/rmt2	non déclenchée	déclenchée
non déclenchée	0.40	0.0
déclenchée	0.46	0.14

5.3 Cycle délibération/exécution

Nous adaptons ici le processus de délibération proposé Section 4.4.2, afin de prendre en compte les problématiques liées à l'intégration de niveaux de ressources.

Si nous avons pu dans le chapitre précédent trouver une solution complète à notre problème, la présence des niveaux de ressources nous empêche d'utiliser le même raisonnement. En effet, l'espace des états globaux devenant infini, il nous est impossible de dégager des états remarquables et encore moins de calculer une méta-politique globale. Pour chercher une solution, nous nous intéressons donc aux méthodes locales. L'objectif ici est de conserver les calculs complets concernant la réalisation de *rmt* ainsi que la conception des macro-actions, mais de passer du calcul complet d'une méta-politique à celui, local, d'un agenda de macro-actions. Cet agenda de macro-action est un arbre renvoyant les macro-actions à exécuter, partant d'un état global initial et est limité par un horizon défini à l'aide de la ressource de temps.

Afin de continuer à traiter des problèmes dont l'exécution n'est à priori pas bornée dans le temps, nous intégrerons explicitement un cycle délibération / exécution, permet-

tant de prolonger dans le temps la résolution du problème posé.

- Intégrer les coûts moyens des politiques aux macro-actions puis au calcul des transitions entre états globaux. (page 103).
- Construire un agenda de macro-actions local donnant les macro-actions les plus intéressantes à exécuter (page 106).
- Définir un cycle délibération / exécution, permettant d'alterner la construction et l'exécution d'un agenda de macro-actions (page 111).

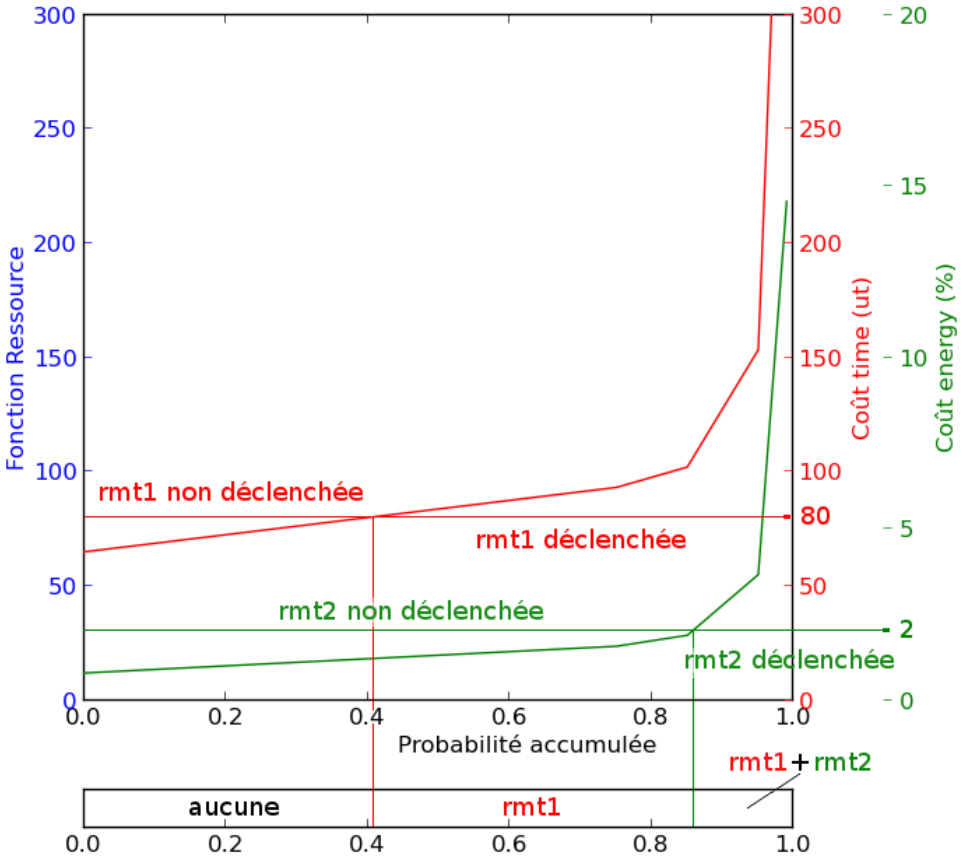


Figure 5.10: Exemple : estimer les combinaisons de déclenchement de plusieurs *rmt* et leur probabilités.
Pour plus de lisibilité, seules les estimations des deux ressources on été conservées, les autres courbes sont masquées.

5.3.1 Calcul de l'effet d'une macro-action

Dans le chapitre 4, nous avons montré comment nous calculons les transitions $[gs, ma, gs']$ et leurs probabilités. En partant de cette base, nous nous intéressons aux rmt dont le déclenchement est conditionné par les ressources ainsi qu'au calcul du niveau de ressource pour les états gs'.

Résoudre les déclenchements de rmt par des ressources

La résolution des effets d'une macro-action ma présentée Section 4.4.2 utilisait la probabilité de terminer par chaque rwt pour déterminer l'état global suivant gs'. Or, les rmt dont le déclenchement est fonction des niveaux de ressources ne sont pas prises en compte. Pour y remédier, nous partons du raisonnement précédent, c'est-à-dire l'analyse des rwt exécutées par la politique, puis nous étudierons les probabilité de déclencher des rmt dépendant des ressources lors de l'exécution. Dans l'impossibilité de prévoir l'état du monde lors du déclenchement d'une rmt dépendant des ressources, nous considérons que le déclenchement d'une telle rmt ne met pas fin à l'exécution de la politique. Dans ce cas, la macro-action déclenche au moins 2 rmt, celle dépendant des ressources, et celle correspondant à la rwt exécutée au moment de terminer l'exécution.

Pour déterminer si une rmt dépendant des ressources se déclenche, nous utilisons la distribution liée des coûts en ressources calculé Section 5.2.2. Ainsi, nous disposons, pour chaque politique et chaque rwt, d'une distribution des coûts permettant de calculer pour toute rmt dépendant des ressources si son seuil de déclenchement est atteint ou non, et sa probabilité. De cette façon, nous dédoublons les scénarios où une rmt dépendant des ressources peut être déclenchée. Une possibilité $[gs, ma, gs'], pr1$ donne alors :

- $[gs, ma, gs'], (pr1 \times (1 - pr2))$
- $[gs, ma, gs''], (pr1 \times pr2)$

Avec $pr2$ la probabilité de déclencher une rmt supplémentaire et gs'' l'état global où cette rmt a été déclenché.

Exemple

Nous poursuivons l'exemple développé Section 4.4.2, dans le chapitre précédent, en changeant l'état de départ. Rappelons donc qu'il s'agit de déterminer les effets :

- de la macro-action ma_{ex}, qui est construite pour recharger le niveau de la batterie une fois, depuis la case E (voir la politique Figure 5.3, page 93). Cette macro-action peut terminer par les rwt suivantes :
  - $rwt1 : (Eu, \mathit{verify}, Eb)$
  - $rwt2 : (Eg, \mathit{recharge}, Eu)$
  - $rwt3 : (Eg, \mathit{recharge}, Eb)$
  - $rwt4 : (C, \mathit{wait}, C)$
  - $rwt5 : (G, \mathit{wait}, G)$
- depuis l'état $gs1 = (ws1, msv1)$, avec :

- $ws1 = (H, emptyHand, noColor, unknown)$, l'état du monde décrivant la situation où le robot est sur la case H, ne porte aucune boîte et où le statut du chargeur est inconnu. Comme pour les autres exemples, nous considérons le cas où la centrale inertielle est imprécise, c'est-à-dire *imuPrecision = imuInaccurate*.
- $msv1 = (faible, jaune1, restreinte, précis)$. Ce *motivation state vector* (msv) décrit respectivement l'état des motivations **Survie**, **Boîtes**, **Zone** et **Relocaliser**, voir Section 3.4.4 pour leurs définitions.

Le tableau suivant donne une partie des scénarios possibles, tel que calculés dans le chapitre précédent. Nous ne traitons pas tous les cas possible, pour éviter les redondances.

transition globale	pr	R	commentaire
$(gs1, ma_{ex}, gs2)$	0.48	+1	avec $rwt2$
$(gs1, ma_{ex}, gs6)$	0.18	0	avec $rwt1$
$(gs1, ma_{ex}, gs6)$	0.05	0	avec $rwt3$
$(gs1, ma_{ex}, gs7)$	0.21	-1	avec $rwt4$

Pour éviter d'alourdir les explications qui suivent, nous utilisons le tableau suivant pour noter les différents état globaux que nous pourrons rencontrer.

gs	ws				msv			
	rP	hS	bC	cS	survie	boîtes	zone	relocaliser
$gs1$	H	emptyHand	noColor	unknown	faible	jaune1	restreinte	précis
$gs2$	E	emptyHand	noColor	unknown	faible	jaune1	restreinte	précis
$gs3$	E	emptyHand	noColor	unknown	bon	jaune1	restreinte	précis
$gs4$	E	emptyHand	noColor	unknown	faible	jaune1	restreinte	imprécis
$gs5$	E	emptyHand	noColor	unknown	bon	jaune1	restreinte	imprécis
$gs6$	E	emptyHand	noColor	bad	faible	jaune1	restreinte	précis
$gs7$	E	emptyHand	noColor	bad	faible	jaune1	restreinte	imprécis
$gs8$	C	emptyHand	noColor	unknown	faible	jaune1	restreinte	précis
$gs9$	C	emptyHand	noColor	unknown	faible	jaune1	restreinte	imprécis

Puisque nous considérons à présent les ressources ainsi que les *rmt* qui en dépendent, de nouveau éléments s'ajoutent au problème :

- le niveau de la ressource de temps initial est : 1 000 **ut**.
- le niveau d'énergie initial est : 32%
- deux *rmt* dépendant des ressources sont présentes lors de l'exécution :
  - *rmtR*, la *rmt* faisant passer la motivation **Relocaliser** de l'état *précis* à l'état *imprécis*. Dans la situation actuelle, cette *rmt* se déclenchera à *time = 1080*.
  - *rmtS*, la *rmt* surveillant le niveau de la batterie pour la motivation **Survie**. Elle fait passer l'état de la motivation de *faible* à *bon* lorsque le niveau d'énergie passe au-dessus de 50%. L'action *recharge* remonte le niveau de la batterie de 20%.

En utilisant les calculs présentés Section 4.4.2, nous pouvons déterminer dans quels cas $rmtR$ et $rmtS$ peuvent être déclenchées ainsi que leur probabilité de déclenchement. Le tableau suivant donne les différentes possibilités prenant en compte les ressources. La Figure 5.10 donne un détail de ce qui a été calculé pour déterminer les scénarios pour $rwt2$ terminal.

transition globale	pr	rwt	commentaire
$(gs1, ma_{ex}, gs3)$	0.192	$rwt2$	$rmtS$ déclenchée
$(gs1, ma_{ex}, gs5)$	0.221	$rwt2$	$rmtS$ et $rmtR$ déclenchées
$(gs1, ma_{ex}, gs4)$	0.067	$rwt2$	$rmtR$ déclenchée
$(gs1, ma_{ex}, gs6)$	0.072	$rwt1$	
$(gs1, ma_{ex}, gs7)$	0.108	$rwt1$	$rmtR$ déclenchée
$(gs1, ma_{ex}, gs7)$	0.178	$rwt4$	
$(gs1, ma_{ex}, gs8)$	0.032	$rwt4$	$rmtR$ déclenchée

Déterminer une estimation moyenne des futurs niveaux de ressources. Une fois que nous avons établi les futurs état globaux et leurs probabilités d'être atteints, nous devons définir quel sera leurs niveaux en ressources. Nous choisissons pour cela, de calculer le niveau en ressources moyen pour chaque scénario en utilisant la distribution liée des coûts. Les estimations étant composées de peu de points, le calcul du niveau moyen des ressources est aisé et rapide. Le tableau ci-dessous indique le futur niveau en ressources pour chaque scénario.

transition globale	pr	rwt	*time*	*energy*
$(gs1, ma_{ex}, gs3)$	0.192	$rwt2$	1072.5	51.0
$(gs1, ma_{ex}, gs5)$	0.221	$rwt2$	1089.2	50.5
$(gs1, ma_{ex}, gs4)$	0.067	$rwt2$	1171.1	47.6
$(gs1, ma_{ex}, gs6)$	0.072	$rwt1$	1062.5	27.3
$(gs1, ma_{ex}, gs7)$	0.108	$rwt1$	1198.4	23.4
$(gs1, ma_{ex}, gs7)$	0.178	$rwt4$	1047.0	30.3
$(gs1, ma_{ex}, gs8)$	0.032	$rwt4$	1203.7	25.7

Calcul de la récompense attendue pour une macro-action. À présent que nous connaissons les différents scénarios possibles et leur probabilité, nous pouvons associer à chacun une récompense. Nous additionnons simplement la récompense calculée avec les rwt terminale et celles des rmt déclenchées par les changements de niveau de ressources. Rappel des récompenses des rmt concernées :

- $rmtR$ a une récompense de -1, puisqu'elle signifie que la précision de l'*imu* devient dangereusement faible.

- $rmtS$ a une récompense $+1$, le niveau de la batterie passant à un niveau jugé correct.

transition globale	pr	rwt	R	(base)	$(rmtR)$	$(rmtS)$
$(gs1, ma_{ex}, gs3)$	0.192	$rwt2$	+2	1		+1
$(gs1, ma_{ex}, gs5)$	0.221	$rwt2$	+1	1	-1	+1
$(gs1, ma_{ex}, gs4)$	0.067	$rwt2$	0	1	-1	
$(gs1, ma_{ex}, gs6)$	0.072	$rwt1$	0	0		
$(gs1, ma_{ex}, gs7)$	0.108	$rwt1$	-1	0	-1	
$(gs1, ma_{ex}, gs7)$	0.178	$rwt4$	-1	-1		
$(gs1, ma_{ex}, gs8)$	0.032	$rwt4$	-2	-1	-1	

Tableau récapitulatif. Pour finir, voici le tableau final décrivant les données des transitions globales examinées pour $(gs1, ma_{ex})$:

transition globale	pr	R	$time$	$energy$
$(gs1, ma_{ex}, gs3)$	0.192	+2	1072.5	51.0
$(gs1, ma_{ex}, gs5)$	0.221	+1	1089.2	50.5
$(gs1, ma_{ex}, gs4)$	0.067	0	1171.1	47.6
$(gs1, ma_{ex}, gs6)$	0.072	0	1062.5	27.3
$(gs1, ma_{ex}, gs7)$	0.108	-1	1198.4	23.4
$(gs1, ma_{ex}, gs7)$	0.178	-1	1047.0	30.3
$(gs1, ma_{ex}, gs8)$	0.032	-2	1203.7	25.7

5.3.2 Construction de l'arbre local des macro-actions

Une fois en possession du calcul des effets d'une macro-action, nous pouvons construire un arbre local des macro-actions. Cet arbre décrit, depuis un état global donné, les futurs états globaux, leurs probabilités et leurs récompenses accumulées R_{sys}, et pour toutes les combinaisons de macro-actions qu'il sera possibles d'exécuter. Le développement de cet arbre sera limité par plusieurs paramètres, et le plus important d'entre eux est un horizon exprimé à l'aide de la ressource de temps.

Construction de l'arbre

Pour construire notre arbre, nous allons nous inspirer d'une représentation courante des réseaux bayésiens, illustrée Figure 5.11. Ainsi, un arbre local des macro-actions sera constitué de :

- **Deux types de nœuds** :
  - **Nœuds états** : il s'agit d'un nœud représentant un couple $(gs, pr, R_{sys}, time)$, un état global gs, une probabilité pr d'arriver à ce nœud, un niveau de récompense global pour cet état, R_{sys}, et la valeur de la ressource de temps de cet état global.
  - **Nœuds macro-actions** : ces nœuds sont présents juste après un nœud état représentent les différentes macro-actions qui peuvent être exécutées.

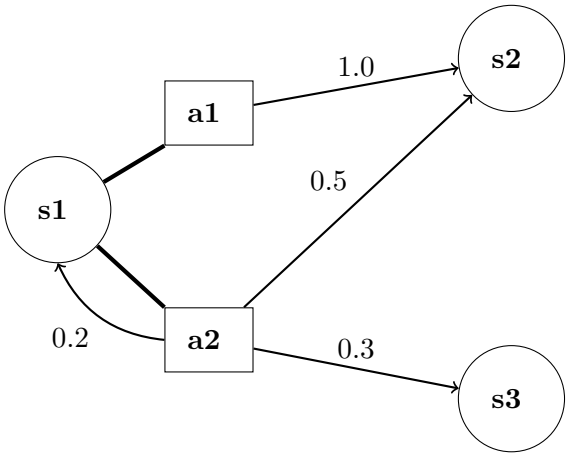


Figure 5.11: Exemple de représentation d'un réseau bayésien.

- **Deux types d'arcs** :
  - **Arcs exécutions** : ils relient un nœud macro-action et un nœud état. Ils représentent une exécution possible de la macro-action ma du nœud macro-action précédent sur l'état global du nœud état précédent. À chaque arc exécution est associé un couple (rwt, pr, r, c), issu possible de l'exécution de ma, avec r la récompense obtenue lorsque la macro-action ma termine avec rwt.
  - **Arcs choix** : ces arcs relient simplement un nœud état aux nœuds des macro-actions disponibles depuis l'état global.

La racine d'un arbre des macro-actions correspond à l'état global au moment où le processus de délibération à commencé. Cet état global commence avec une probabilité $pr = 1$, $R_{sys} = r$ et $time = t$ **ut**. À partir de chaque nœud, nous développons l'arbre pour connaître les états globaux qu'il est possible d'atteindre, la probabilité et le temps (en unité de temps **ut**) nécessaire pour les atteindre ainsi que le niveau de récompense pour ces futurs états. Toutes les macro-actions ma *disponibles* (voir définition Section 4.2.1) depuis gs sont développées en calculant l'effet de ma sur gs. Les calculs des effets de ma doivent être effectués lors de la délibération, puisqu'ils dépendent du niveau des variables continues (i.e. les ressources). De nouveaux nœuds apparaissent alors dans l'arbre, comme illustré dans la Figure 5.12.

Paramètres limitant le développement

L'arbre des macro-actions peut être développé sans limite, nous devons donc définir des paramètres qui limitent son développement.

- **Horizon** (exprimé en **ut**) : il s'agit du paramètre limitant principal. L'horizon H est fixé en ajoutant une valeur fixe à la valeur de la ressource *time* de l'état global initial. Tout état gs dont la valeur de la ressource *time* est supérieur à H ne sera pas développé. Cet horizon a pour objectif de créer un ensemble de scénarios dont l'exécution est de même longueur, pour ensuite pouvoir déterminer le plan maximisant la récompense obtenue par unité de temps.

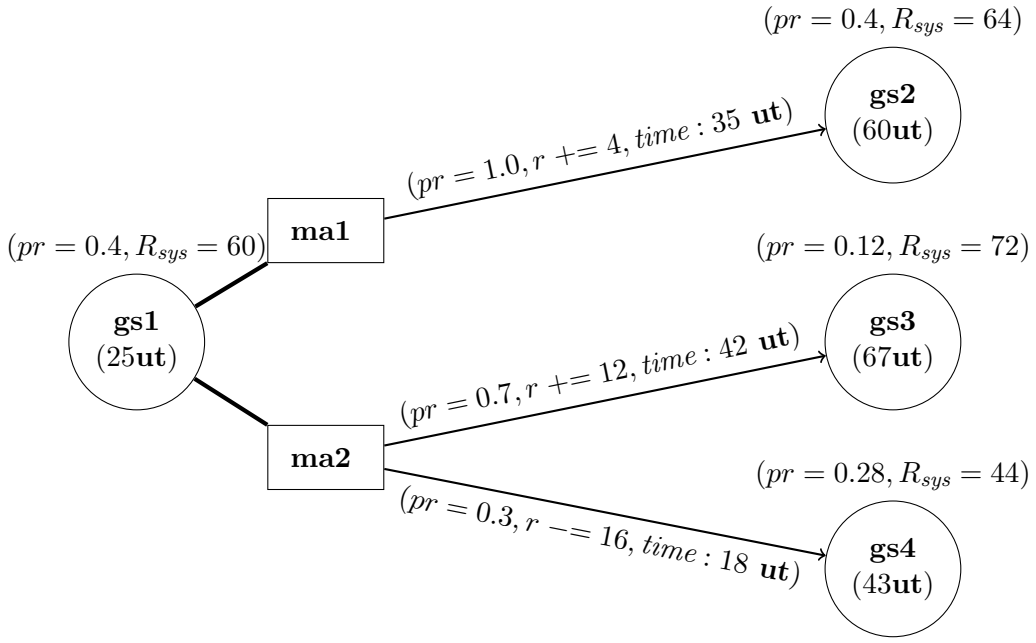


Figure 5.12: Exemple de représentation d'un arbre des macro-actions.

- **Probabilité minimale** : nous pouvons exclure les scénarios les moins probables en déterminant une valeur pr_{min}. Si la probabilité d'un nœud est inférieure à pr_{min}, le nœud ne sera pas développé. Éviter de calculer les états peu probables permet de limiter le coût pour générer un arbre des macro-actions tout en conservant la majorité des scénarios les plus probables.

- **Profondeur maximum** : certain problèmes ou leur modélisation peut présenter des situations où l'agent peut effectuer sans fin des macro-actions très peu coûteuses en temps. Imaginons par exemple si, dans notre exemple récurrent, le robot avait la possibilité de relancer une relocalisation sans attendre (dans notre exemple, il doit attendre 720 **ut**). Dans ce cas, l'arbre générerai une grande quantité de scénarios qui enchaînerai majoritairement la macro-action consistant à se relocaliser.

  Dans certains cas comme celui-ci, il peut être préférable de limiter la profondeur de l'arbre des macro-actions. Nous définissons h_{max}, la profondeur maximale atteinte par l'arbre. Noter que cette limitation est spécifique au problème traité.

- **Temps d'exécution** : il peut enfin être intéressant de limiter le développement de l'arbre en limitant le temps de calcul de celui-ci. Nous définissons alors une durée en seconde pendant lequel le calcul se déroule avant de s'arrêter. De cette manière, le paramètre de limitation de l'horizon H devient dynamique et le prochain nœud développé est celui dont la ressource *time* est la plus faible.

  Cette méthode a pour avantage de s'exécuter en un temps fixe, permettant de mieux tenir compte du temps passé à calculer un agenda et de connaître donc son impact sur l'évolution du monde.

Exemple de développement

Nous illustrons rapidement le développement d'un court arbre des macro-actions à l'aide de la Figure 5.13. Nous commençons ici avec avec $gs1$ comme état global de départ. Les paramètre limitant le développement de l'arbre sont :

- **Horizon** = 60 **ut**.
- **Probabilité minimal** = 0.05.
- **Profondeur maximale** = 3.

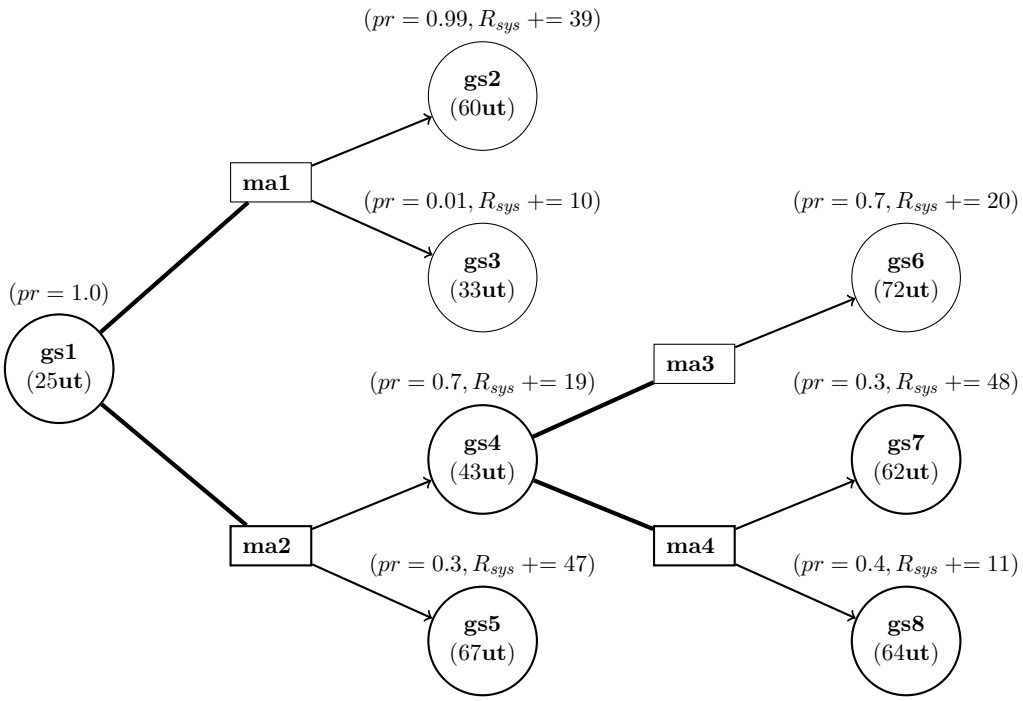


Figure 5.13: Arbre des macro-actions exemple.

Nous commençons par développer $gs1$, et donc par demander au module intentionnel la liste des rmt disponibles depuis $gs1$. Nous récupérons ensuite les macro-actions correspondantes $ma1$ et $ma2$, qui ont été constituées par le module délibératif et à l'aide des calculs effectué par le module opérationnel. Nous les plaçons dans l'arbre comme fils de $gs1$, puis déterminons les états globaux dans lesquels ces macro-actions peuvent mener.

Nous calculons donc les effets de $ma1$ lorsqu'elle est exécutée à partir de $gs1$, et obtenons les deux possibilités suivantes :

- **gs2** : avec une probabilité de 0.99 et une récompense de 39, pour une durée de 35 **ut**.
- **gs3** : scénario peu probable avec une probabilité de 0.01, une récompense de 10 et une durée de 8 **ut**.

Nous plaçons deux nœuds correspondants dans l'arbre. Le niveau de la ressource de temps des nœuds fils est égal à la somme de celui du nœud père et de la durée de la macro-action pour aller du père au fils. La probabilité du nœuds fils est égale à la multiplication entre la probabilité du nœud père (avec $pr = 1.0$ pour la racine) et la probabilité que la macro-action mène du nœud père au nœud fils. Nous effectuons ensuite la même construction pour $ma2$ avec les possibilités suivantes :

- **gs4** : scénario consommant peu de temps, 18 **ut**, avec une probabilité de 0.7 et une récompense de 10.
- **gs5** : avec une probabilité de 0.3, une récompense de 47 et une durée de 42.

À présent que $gs1$ est développé, nous regardons lesquels de ses nœuds fils vont être développés à leur tour :

- **gs2** : ce nœud a atteint l'horizon de l'arbre, il n'est donc pas développé.
- **gs3** : ce nœud n'a pas atteint l'horizon. Cependant, la probabilité d'atteindre ce nœud est inférieure à la probabilité minimal définie. Pour éviter de calculer les scénarios trop peu probables, il n'est pas développé.
- **gs4** : la probabilité de ce nœud est satisfaisante, et il se situe avant l'horizon. Sa profondeur est inférieure à la profondeur maximum définie, **il est donc développé**.
- **gs5** : idem que pour $gs2$.

Nous développons ensuite $gs4$ de la même manière que $gs1$. Nous ne détaillons pas une seconde fois le développement, la Figure 5.13 donne l'arbre finit. Les nœuds des états $gs6$, $gs7$ et $gs8$ ne sont pas développés car ils dépassent l'horizon fixé.

5.3.3 Calcul d'un agenda de macro-actions

Une fois que nous avons construit un arbre des macro-actions, nous connaissons donc les différentes possibilités qui s'offrent à l'agent pour un horizon fixé. Nous allons à présent calculer l'*agenda de macro-actions*, et donc choisir, pour chaque nœud ayant des fils, la macro-action la plus prometteuse, c'est-à-dire celle maximisant les gains pour R_sys.

Pour cela, nous allons, en partant des feuilles de l'arbre, choisir la macro-action maximisant l'espérance de la récompense obtenue pour l'horizon fixé. L'espérance de la macro-action ma, E_{ma}, est calculée comme suit :

$$E_{ma} = \sum r(gs, ma, gs')$$
$$r(gs, ma, gs') = \frac{(H - t(gs)) \times pr(gs') \times r_{sys}(gs')}{max(t(gs') - t(gs), H - t(gs)) \times pr(gs)}$$

Pour éviter que les nœuds dont le niveau de la ressource de temps dépassent l'horizon soient favorisés, nous réduisons au prorata du temps dépassé la récompense espérée si gs' dépasse l'horizon. Le nœud $gs6$ de notre exemple l'illustre bien : celui-ci est atteint pour $time = 72$ alors même que l'horizon H est fixé à $time = 60$.

Une fois la meilleure macro-action sélectionnée pour un état gs, son espérance de récompense est ajoutée à $r_{sys}(gs)$ et l'algorithme continue. Cette méthode permet de garantir que nous maximisons l'espérance de la récompense qu'il est possible d'obtenir pour l'horizon H fixé. Pour finir, nous obtenons un arbre pour lequel chaque état global

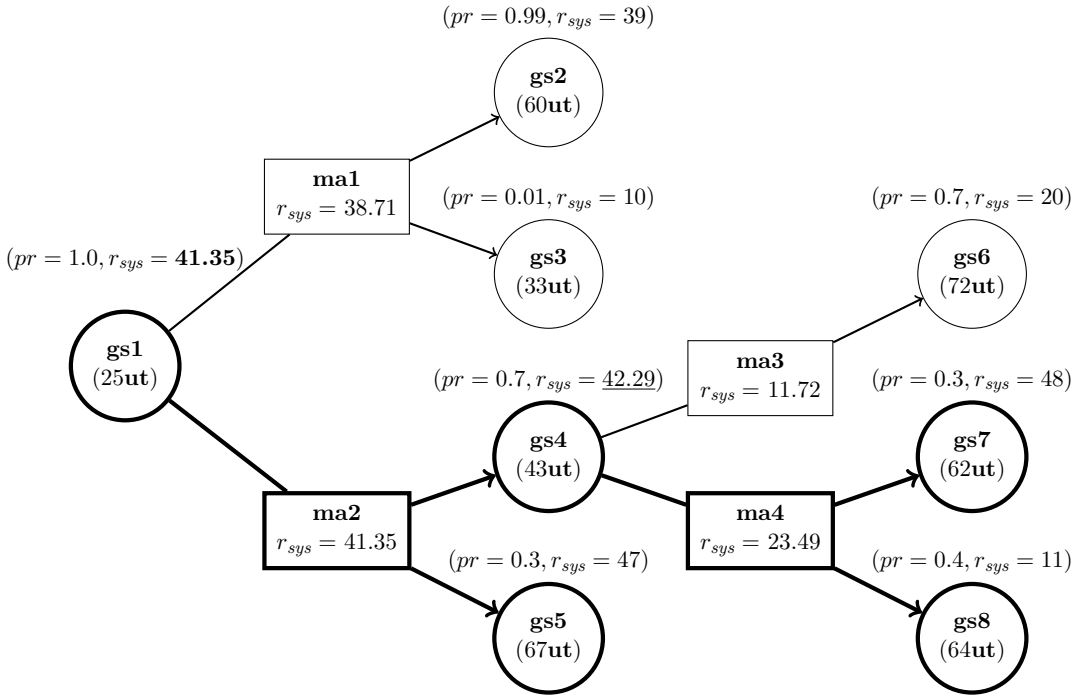


Figure 5.14: Exemple de calcul d'un agenda de macro-actions.
Les éléments en gras sont les éléments conservés pour définir l'agenda, les autres sont supprimés.

est associé la macro-actions maximisant l'espérance de la récompense du système. Les macro-actions qui ne sont pas choisies sont supprimées ainsi que les états auxquels elles mènent. La Figure 5.14 illustre le calcul de l'agenda de macro-actions pour le précédent arbre des macro-actions.

5.3.4 Méthode d'exécution d'un agenda de macro-actions

Une fois calculé par le module délibératif et transmit au module superviseur, un agenda de macro-action est exécuté. Pour cela, le module superviseur doit exécuter les macro-actions de l'agenda en partant de la racine de l'agenda.

Au début de l'exécution de l'agenda, le superviseur commence à exécuter la macro-action correspondant à la racine de l'agenda. Pour cela, il demande au module opérationnel la politiques correspondant à la macro-action à exécuter (le module délibératif et l'agenda de macro-actions utilisent seulement les macro-actions, modèles d'exécutions des politiques π^{msv}_{rmt}). Il envoie donc au système sensori-moteur les actions à exécuter en fonction de l'état courant de l'agent et informe le module délibératif de sa progression. Dès qu'une *rwt* est atteinte, l'exécution de π^{msv}_{rmt} se termine et le module superviseur détermine le nouvel état courant de l'agenda à l'aide de l'état actuel et de la *rwt* exécutée, puis commence l'exécution de la nouvelle macro-action.

Si le superviseur atteint la fin de l'agenda (c'est-à-dire une feuille de l'arbre), il l'annonce au module délibératif et attend la constitution d'un nouvel agenda à exécuter. Il peut également arriver que, à la fin de l'exécution d'une politique, l'état global courant

ne soit pas présent dans l'arbre. Les estimations faites sur la consommation des ressources étant approximative, il peut arriver certain cas où la consommation en ressource a été supérieur aux cas estimés. Dans ce cas, des motivations peuvent changer d'état sans que cela ait été calculé par le module délibératif. L'arbre actuel est abandonné et nous en construisons un nouveau commençant dans l'état actuel.

5.4 Conclusion

Au cours de ce chapitre, nous avons adapté la solution au problème de décision probabiliste avec des objectifs multiples et permanents du chapitre précédent pour y incorporer le traitement de ressources. Nous avons défini une ressource r comme une variable numérique continue, dont la valeur est modifiée par les actions exécutées par le robot, à l'aide de la fonction de coût $C_r(ws, a, ws')$.

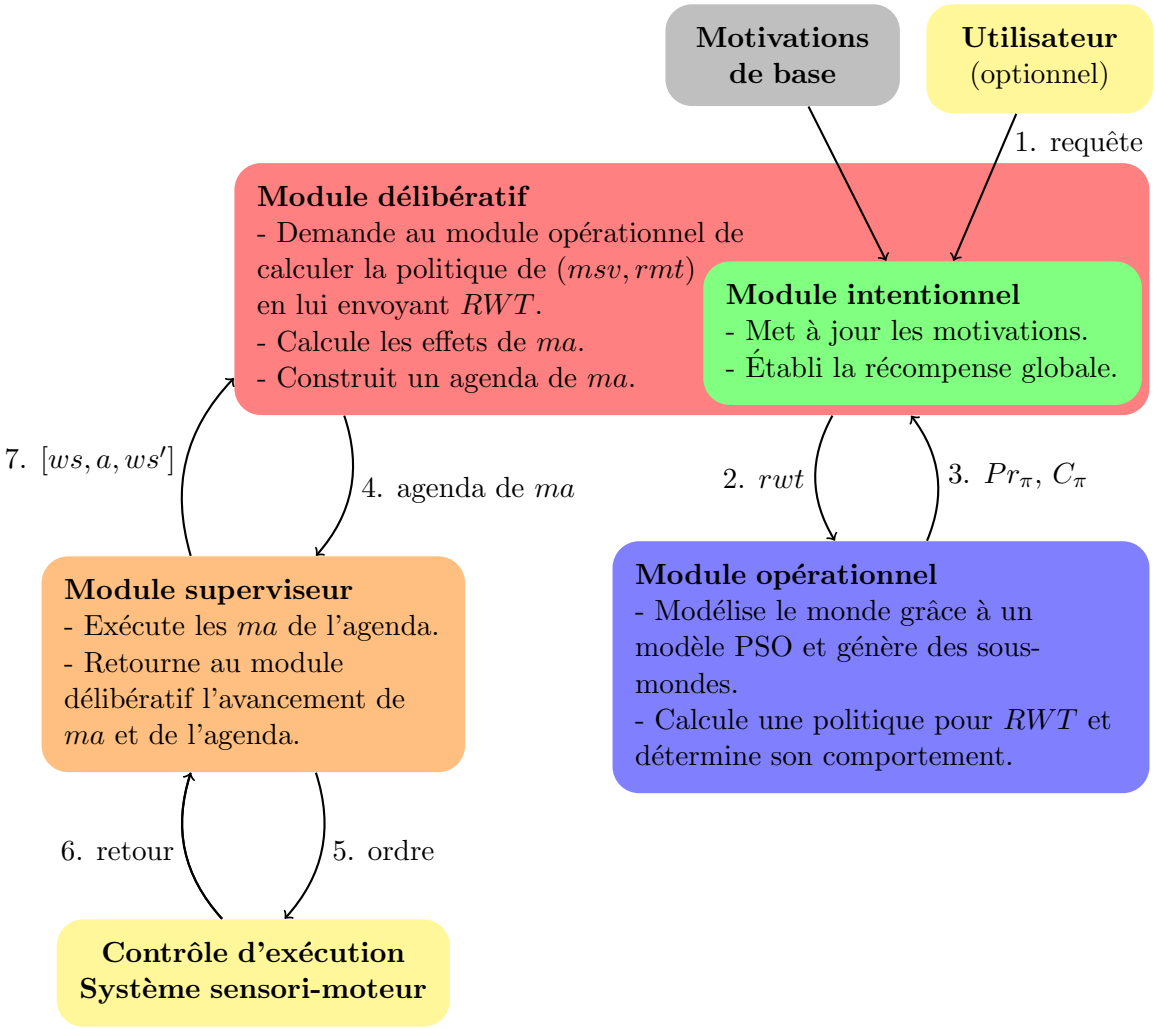


Figure 5.15: Architecture complète (version avec ressources), composée des modules opérationnel, intentionnel et délibératif.

Pour incorporer la problématique des ressources et du traitement des variables con-

tinues qui les représentent, nous avons choisi de les considérer à part. Les niveaux des ressources sont uniquement modifiés par les actions exécutées, mais l'exécution des actions n'est pas influencée par les ressources. Nous avons alors cherché une méthode afin de connaître le modèle de la consommation des macro-actions afin de prévoir le déclenchement des *rmt* conditionnées par des niveaux de ressource. En l'absence de méthode exacte utilisable, nous avons choisi d'utiliser une méthode inexacte donnant une estimation rapide.

Si le problème résolu dans le chapitre précédent contenait un nombre fini d'état globaux gs, l'apparition des variables continues modélisant les ressources nous empêche de considérer tous les états possibles et de calculer une politique globale. Nous avons donc choisi d'opter pour une recherche locale, en constituant un agenda de macro-actions, un plan sous forme d'arbre, limité dans le temps et donnant les macro-actions à exécuter pour maximiser la récompense du système R_{sys}. En conséquence, l'architecture est modifiée pour prendre en compte ces modifications et est décrite Figure 5.15. Cette architecture prend en compte un cycle délibération / exécution, permettant résoudre un problème contenant des ressources par l'exécution en chaîne d'agendas de macro-actions.

Chapitre 6

Résultats expérimentaux

Contents

Pour tester les performances et les possibilités de notre architecture, nous nous sommes intéressés à un problème de type *fetch-and-carry* généralisé à l'assemblage, où le robot effectue un ensemble de tâches répétitives, de navigation, de transport d'objets et de fabrication, dont nous nous sommes inspirés et que nous avons simplifié pour l'exemple récurrent utilisé dans les chapitres précédents. Ces tests ont été réalisés en simulation.

6.1 Problème évalué

Le problème que nous avons choisi de traiter prend place dans un atelier, dans lequel notre agent, un robot manufacturier autonome, se déplace et construit des ventilateurs qu'il doit ensuite placer sur un tapis roulant. Il doit se maintenir en état de fonctionner et se recharger à l'aide de bornes de rechargement à sa disposition. Cependant, il arrive que les bornes dysfonctionnent, le robot doit alors suivre une procédure pour les remettre en marche. De plus, une partie de l'atelier lui est périodiquement restreinte et il doit donc éviter d'y pénétrer.

Nous avons choisi ce problème pour un ensemble de propriétés :

- **Pas de fin explicite** : la situation présentée est idéale pour un scénario n'ayant pas de fin. le robot travaille indéfiniment à sa tâche.
- **Multiples objectifs conflictuels** : les objectifs proposés entrent tous en conflit les uns avec les autres (même si la zone périodiquement restreinte ne demande aucune action). Le robot doit donc choisir lui-même comment utiliser le temps et l'énergie dont il dispose pour accumuler le plus de récompense possible.

  La fabrication de ventilateurs utilise de l'énergie et risque de faire passer le robot par la zone restreinte. La recharge de la batterie prend du temps, empiétant sur le temps alloué à la construction et risque également de faire passer par la zone restreinte. Remettre en marche les bornes de chargement pose également le même problème, mais est en plus indispensable pour pouvoir charger à nouveau la batterie.
- **Prise en compte des ressources** : Le temps et l'énergie sont modélisés comme décrit dans le Chapitre 5. Ces deux ressources ne sont pas uniquement utilisées comme des contraintes ou des facteurs d'optimisation, elles ont un rôle dans les objectifs fixés. L'objectif du maintien du niveau de la batterie fait directement référence à la ressource pour savoir si la batterie doit être rechargée et définir son niveau d'urgence. Le temps est utilisé pour définir la zone restreinte d'accès et, nous le verrons plus tard, pour pénaliser le robot s'il reste trop longtemps avec un niveau de batterie faible.

6.1.1 Définition du problème

Le robot est placé dans un atelier fermé, découpé en zones et peut se déplacer d'une zone à l'autre. Certaines portes menant d'une pièce à une autre peuvent être chacune fermés ou ouvertes et le robot ne peut ni les ouvrir, ni les fermer. L'état des portes change de façon inopinée.

Dans cet environnement, le robot doit fabriquer des ventilateurs à partir de pièces détachées, présentes en grand nombre dans une pièce de stockage. Il doit pour cela effectuer deux opérations : assembler un cadre et un moteur, puis ensuite y ajouter des pales. Pour la construction, le robot doit être sur un poste de travail qui lui fournit les outils

nécessaires. Il y a deux postes disponibles dans l'atelier, un bon poste (où le travail est facile) et un mauvais poste (où le travail est difficile, et peut échouer). Toutes les opérations d'assemblages peuvent être effectuées indifféremment sur chaque poste. Le ventilateur construit doit être déposé sur un tapis roulant, qui l'emmène en dehors de l'environnement du robot.

Le robot doit également rester en état de fonctionnement et donc ne pas être à court d'énergie. Il dispose pour cela de deux bornes de rechargement. Il peut choisir de se recharger rapidement ou lentement, changeant la quantité d'énergie reçue. Cependant, lorsqu'il se recharge, il y a un risque que les chargeurs dysfonctionnent. Dans ce cas, le robot ne reçoit pas d'énergie et les deux bornes sont désactivées. Pour les remettre en marche, le robot doit alors identifier la cause du problème, qui peut être lié aux chargeurs ou à l'alimentation générale. Dans le premier cas, le robot peut simplement réactiver les bornes (à partir de n'importe laquelle, et réactivant les deux bornes). Dans le second cas, le robot doit d'abord aller effectuer une opération sur le panneau d'alimentation, puis ensuite réactiver les bornes comme dans le premier cas.

Enfin, une zone de l'atelier est périodiquement restreinte et le robot doit éviter d'y pénétrer. La zone en question comporte une borne de chargement, le bon poste de fabrication, ainsi que le tableau électrique

La Figure 6.1 illustre la représentation du problème que nous utilisons pour suivre les progrès lors des expérimentations.

Fonctionnement. Nous décrivons ici comment l'environnement du robot évolue en fonction des actions qu'il réalise :

- **Déplacements** : le robot se trouve dans l'une des quinze zones de l'atelier et peut se déplacer d'une zone vers une zone adjacente si elle n'est pas bloquée par une porte close. Il y a dans la configuration du problème choisi deux portes pouvant être ouvertes ou fermées, les autres portes restent toujours ouvertes. À cause de l'imprécision des déplacements, le robot peut échouer lorsqu'il essaie de se déplacer. Lors d'un échec, il peut rester sur place ou arriver par erreur sur une autre case adjacente. Que les zones soient vides ou pleines (poste, borne, etc.) n'a pas d'influence sur les déplacements. Le robot peut activer ses freins, ce qui l'empêche de se déplacer, mais lui permet d'utiliser par exemple les postes de fabrication.

- **Objets** : les cinq objets (le cadre, le moteur, le cadre et le moteur assemblés, les pales et le ventilateur complet) peuvent être soit déposées dans un des lieux de stockage, soit être placés sur la machine d'assemblage, soit dans une des mains du robot (gauche, droite). Le robot a deux mains et peut ramasser ou déposer des objets. Il peut aussi déposer un ventilateur (complet uniquement) sur le tapis roulant. Celui-ci sort et disparaît ensuite de l'environnement de l'agent. Le robot peut enfin récupérer librement chacun des trois objets de base (cadre, moteur et pales) depuis l'un des stockages. Nous considérons qu'il y en a une quantité illimitée à disposition.

- **Assemblage** : pour assembler des ventilateurs, le robot doit d'abord apporter un cadre et un moteur à un poste de fabrication. Le résultat doit ensuite être associé à des pales pour constituer un ventilateur. Le bon poste de fabrication permet au robot de réussir l'assemblage à chaque fois, ce qui n'est pas le cas sur le mauvais

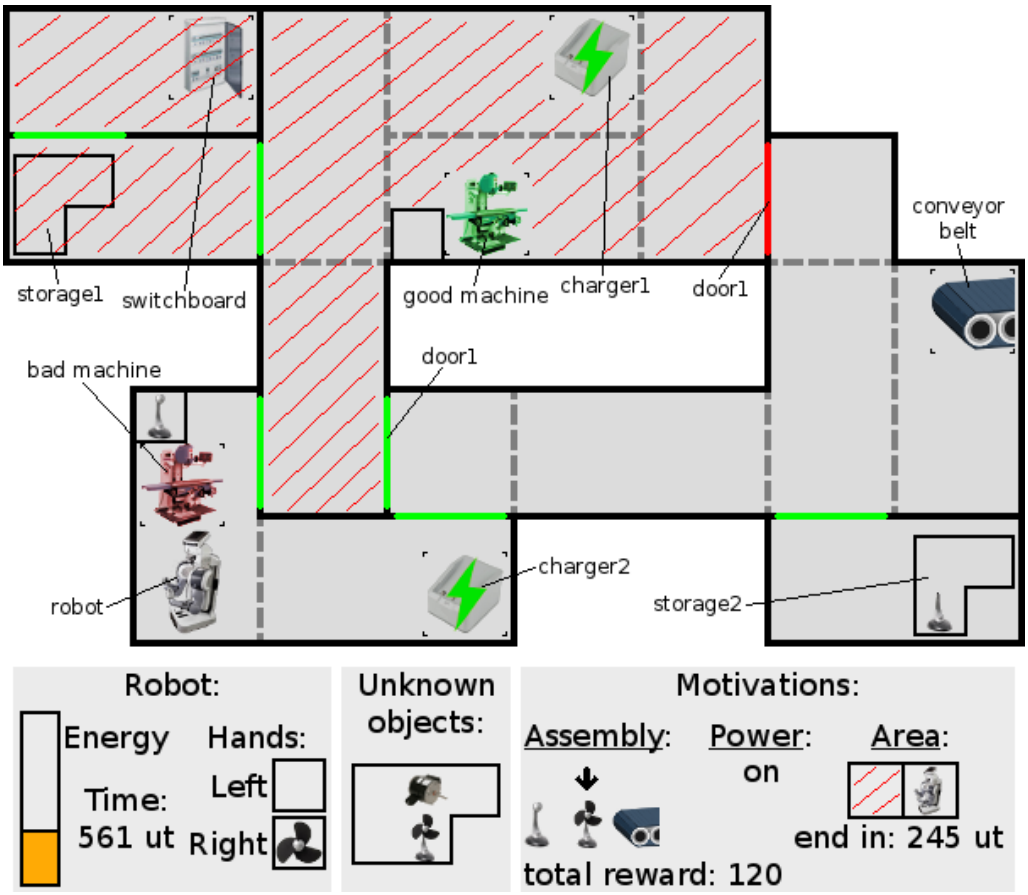


Figure 6.1: Représentation graphique du problème.
L'environnement, la position du robot et la positions des objets présents sont représentés dans la partie supérieure. Le tableau électrique, les machines pour assembler les ventilateurs et le tapis roulant sont représentés dans leurs emplacement respectifs. La partie de l'environnement hachurée correspond à la zone pouvant être restreinte. L'état de la batterie est donné par la jauge en bas à gauche, et le temps passé depuis le début de l'expérience est indiqué à sa droite. Les objets dont la position est inconnue (pas encore fabriqués par exemple) sont placé en dessous. Les motivations et leurs états sont indiqués en bas à droite, ainsi que la quantité de récompense accumulée depuis le début de l'expérience. Noter que la jauge d'énergie sur la gauche suffit pour la motivation **Survie**, et "Power" désigne l'état de fonctionnement des chargeurs. Le temps passé depuis le début de l'expérience et le temps avant que la zone soit à nouveau restreinte (ou libre) sont exprimés en unité de temps **ut**.

poste. En cas d'assemblage réussi, les objets ayant servi disparaissent et l'objet résultant apparaît sur le poste.

- **Rechargement** : lorsqu'ils sont activés, les bornes de chargement permettent au robot de recharger sa batterie. Lors d'un rechargement, il y a un possibilité pour qu'un problème survienne, et les bornes (les deux en même temps) se désactivent. La cause d'un dysfonctionnement est aléatoire, le robot doit alors l'identifier en examinant une des bornes. Dans un cas, réactiver les bornes suffit. Dans un second, le robot doit d'abord effectuer une opération sur le tableau électrique, avant de réactiver les bornes.

- **Coûts en ressources** : chaque action a un coût en ressource propre et qui dépend du résultat de l'action. Par exemple, nous considérons que le robot utilise plus de temps et d'énergie lorsqu'un déplacement échoue (il reste alors sur place ou arrive sur une autre case que celle souhaitée), que lorsqu'il réussit. Cela est d'autant plus évident lorsque le robot se recharge : il ne récupère pas d'énergie lorsqu'un dysfonctionnement se produit.

Objectifs. Le problème comporte quatre objectifs distincts et concurrents. Ces objectifs sont toujours présents, mais certains ne sont actifs que lors de certaines périodes ou dans des états du monde précis. De plus, un des objectifs ne demande la réalisation d'aucune tâche. Nous présentons à présent en détail chacun d'entre eux :

- **Assemblage** : consiste à produire des ventilateurs puis à les déposer sur le tapis roulant. Cet objectif est divisé en trois tâches cycliques : assembler un cadre et un moteur, puis rajouter les pales, et enfin déposer le ventilateur terminé sur le tapis roulant. Le robot est récompensé pour chaque tâche réussie. Si une pièce est cassée (l'assemblage cadre / moteur par exemple), le robot subit une pénalité et doit recommencer les étapes invalidées.

- **Survie** : force le robot à conserver un bon niveau d'énergie. Consiste à établir quatre intervalles de niveaux d'énergie, allant de *complet* à *critique*. Dans tous les niveaux inférieurs à *complet*, la tâche consistant à recharger sa batterie est activée, et la récompense est d'autant plus grande que le niveau de la batterie est faible. Dans l'intervalle *complet*, il n'y a aucune tâche de disponible, le robot n'a donc rien à faire pour cet objectif. Lorsque le niveau de la batterie passe d'une intervalle à une intervalle inférieure, le robot reçoit une pénalité, d'autant plus forte que le niveau d'énergie est bas. De plus, il reçoit périodiquement une pénalité lorsqu'il se trouve dans l'intervalle *critique*.

- **Maintenance** : consiste à maintenir les bornes de chargement de la batterie en fonctionnement. Cet objectif n'a pas de tâche disponible lorsque les chargeurs fonctionnent. En cas de dysfonctionnement, le robot doit effectuer la procédure pour remettre en état les chargeurs, permettant d'obtenir une récompense.

- **Zone** : restreint périodiquement l'accès à une zone de l'atelier. L'alternance entre l'état restreint et l'état libre est fixe et connu à l'avance. Lorsque la zone est restreinte, le robot reçoit une pénalité s'il pénètre dans la zone, volontairement ou non.

6.1.2 Modèle du problème

À partir de la description du problème, nous allons proposer une modélisation pour pouvoir le résoudre à l'aide de l'architecture que nous avons proposé dans ce travail.

Il s'agir tout d'abord de construire le modèle opérationnel, c'est-à-dire un modèle MDP sans fonction de récompense, permettant à un planificateur de calculer une politique pour toutes les tâches que le robot peut avoir à accomplir pour satisfaire ses objectifs. Nous entendons ici par tâche, toutes séries d'action permettant une avancée dans la satisfaction d'un objectif. Il peut s'agir d'aller réaliser l'action consistant à se recharger, ou de réaliser la pièce intermédiaire en assemblant un cadre et un moteur.

Ensuite, et en nous appuyant sur le modèle opérationnel, nous traduisons les objectifs définis en *motivations*. Il nous faut dégager les différentes tâches à exécuter par le robot et les organiser logiquement au sein d'un automate.

Modèle opérationnel

Nous donnons ici le modèle opérationnel que nous avons choisi pour décrire les actions que le robot peut réaliser. Par soucis de lisibilité, certaines données sont volontairement simplifiées.

L'état du monde est constitué de sept variables d'état. Nous rappelons qu'un état du monde est composé d'une valuation pour chacune de ces variables d'état. Notons que l'état de la zone (restreinte ou non) n'est pas indiquée car si elle aura une influence sur le choix d'éviter ou non la zone, elle n'influence en rien la capacité du robot à se déplacer dans la-dite zone. Les variables d'état sont :

- **Position du robot** : les quinze positions possibles pour le robot, notée de "A" à "O".

- **Portes** : les portes *porte1* et *porte2* peuvent être chacune soit ouverte soit fermée. Pour que leurs états change de temps en temps, nous avons choisi de les ouvrir ou de les refermer aléatoirement lorsque le robot dépose un ventilateur sur le tapis roulant.

- **Freins** : dans l'état *on* lorsque les freins sont activés, dans l'état *off* sinon.

- **Position des objets** : un objet peut se trouver soit dans une des mains du robot, soit dans un des stockages, soit sur un poste de fabrication lorsqu'il vient d'être assemblé, ou enfin *indisponible* si il a été consumé ou détruit. Les objets sont : *cadre*, *moteur*, *pales*, *cadre&moteur* and *ventilateur*. Dans le modèle utilisé, chaque objet possède sa propre variable d'état. De plus, chaque type d'objet n'est représenté qu'une seule fois dans le système. Une fois consommé ou détruit, l'objet peut à nouveau être récupéré dans une des deux réserves.

- **Chargeurs** : les chargeurs peuvent être dans l'état *activés* s'ils sont en état de fonctionnement normal, *problème* s'ils subissent un dysfonctionnement et qui n'a pas encore été identifié, *désactivés* s'ils ont simplement besoin d'être réactivés, ou *nonAlimentés* s'il est nécessaire de rétablir leur alimentation à l'aide du tableau électrique.

Les ressources du système sont :

- **Énergie** : une ressource représentée par une valeur numérique entre 0 et 100, et qui représente le pourcentage restant de la batterie du robot (100% représente une batterie pleine, 0% une batterie vide).

- **Temps** : une valeur numérique représentant la ressource du temps qui s'écoule. Sa valeur commence à 0 et s'exprime en unité de temps **ut**.

L'action que le robot peut exécuter sont listées ici. Nous précisons pour chaque action la liste des variables d'état nécessaires pour exécuter l'action, que ce soit pour les conditions ou les effets. Ces variables d'état sont indiquées entre crochets.

- **Déplacer(direction)** [position du robot, portes, freins] : le robot peut essayer de se déplacer de la position dans laquelle il se trouve vers une position adjacente. Il peut se déplacer dans quatre directions : à gauche, à droite, en haut et en bas. Cette action n'est possible que si les freins du robot sont désactivés et si la position adjacente visée existe et n'est pas bloquée par une porte. Un déplacement a une probabilité de succès de 0.70, une probabilité de 0.20 de ne pas se déplacer et 0.10 d'arriver sur une autre case adjacente. Si il n'y a aucune autre case adjacente, les chances de succès sont de 0.80.

- **Freiner** [freins] : le robot peut activer ses freins ou les désactiver lorsqu'ils sont déjà enclenchés. Cette action réussit toujours. Avoir les freins activés empêche de se déplacer, mais permet d'utiliser les bornes de rechargement ainsi que les postes d'assemblage. De plus, cela améliore les chances de succès lorsque le robot essaye d'attraper un objet.

- **Recharger(quantité)** [position du robot, freins, chargeurs] : lorsque le robot souhaite recharger le niveau de sa batterie, il peut essayer d'en recharger 10% ou 25% selon l'argument passé à l'action. Pour ces deux arguments, l'action possède un taux de succès respectivement de 0.96 et de 0.90. En cas de succès, la batterie est rechargée. En cas d'échec, les chargeurs passent dans l'état *problème*. Pour effectuer l'action, le robot doit être devant un chargeur en état de fonctionnement et avoir ses freins d'activés.

- **Prendre(main,objet)** [position du robot, freins, position des objets] : le robot peut essayer de prendre un objet présent sur la même position que lui si sa main est libre. Cette action a un taux de succès de 0.80 et de 0.90 si les freins du robot sont activés. En cas d'échec, rien ne se passe.

- **Récupérer(objet)** [position du robot, freins, position des objets] : si l'un des trois objets de base (*cadre*, *moteur* et *pales*) est dans l'état *indisponible*, il peut être récupéré par le robot. Il doit pour cela se trouver dans un des deux stockages de l'atelier. Cette action réussit toujours.

- **Assembler** [position du robot, freins, position des objets] : permet au robot d'assembler les deux objets qu'il tient dans ses mains. Cette action doit être exécutée depuis un poste de fabrication, avec les freins activés et deux objets dans les mains. Il existe deux formules valides : *cadre* et *moteur* donnent *cadre&moteur*, *cadre&moteur* et *pales* donnent *ventilateur*. Le bon poste de fabrication offre une probabilité de réussite de 1.00, et le mauvais poste une probabilité 0.60. En cas de succès, les deux objets dans les mains du robot deviennent *indisponibles* et l'objet créé apparaît sur le poste de fabrication. Lors d'un échec, il y a une probabilité de 0.40 qu'un des deux objets initiaux se casse (0.20 chacun), 0.10 que les deux objets soient détruits. Dans le reste des cas, rien ne se passe.

- **Déposer(ventilateur)** [position du robot, freins, position des objets] : le robot dépose un ventilateur sur le tapis roulant. Le robot doit être placé devant le tapis roulant, les freins activés et avec un ventilateur dans une de ses mains. Cette action réussit toujours et le ventilateur devient alors *indisponible*. De plus, les portes *porte1* et *porte2* ont chacune une probabilité de 0.50 de changer d'état.

action	resource	success	failure
se déplacer	time	+7	+14
	energy	−0.3%	−0.5%
freiner	time	+2	
	energy	−0.1%	
se recharger	time	+20 / +40	+1
	energy	+10% / +25%	−0.1%
prendre	time	+8	+11
	energy	−0.2%	−0.4%
récupérer	time	+6	
	energy	−0.6%	
assembler	time	+20	+28
	energy	−0.3%	−0.4%
déposer	time	+4	
	energy	−0.2%	
identifier	time	+10	
	energy	−0.3%	
réalimenterChargeur	time	+4	
	energy	−0.3%	
activerChargeurs	time	+2	
	energy	−0.1%	

Table 6.1: Table des coûts en ressources des actions du robot. Les coûts peuvent dépendre de la réussite des actions.
Les coûts présentés sont indicatifs, puisqu'il s'agit d'une version légèrement simplifiée du modèle utilisé pour les expérimentations.

- **Identifier** [position du robot, chargeurs] : permet de découvrir la cause du dysfonctionnement des chargeurs. Le robot doit être placé à côté de l'un des chargeurs et leurs statut doit être *problème*. L'état des chargeurs a une chance sur deux de devenir *désactivé* ou *nonAlimentés*.

- **RéalimenterChargeurs** [position du robot, chargeurs] : réalimente les chargeurs en énergie. Le robot doit être placé devant la tableau électrique et le statut des chargeurs doit être *nonAlimentés*. Cette action réussit toujours et passe le statut des chargeurs à *désactivés*.

- **ActivaterChargeurs** [position du robot, chargeurs] : remet les chargeurs en route. Cette action doit être effectuée devant l'un des chargeurs (l'action réactive les deux) et les bornes de chargement doivent être dans l'état *désactivés*. Cette action ne peut rater et l'état des chargeurs devient *activés*.

Pour finir sur la définition du modèle du monde utilisé, nous présentons le Tableau 6.1, qui donne les ressources utilisées par chaque action. Celles-ci dépendent de la réussite ou non de l'action.

Modèle intentionnel

Nous devons à présent modéliser les quatre objectifs définis précédemment, **Assemblage**, **Survie**, **Maintenance** et **Zone**. Pour cela, nous utilisons les *motivations* définies dans ce mémoire. Pour cela, nous utilisons leur description pour définir les états et les *rewarded motivation transitions* nécessaire pour les retranscrire sous la forme d'automates. Les motivations ayant été simplifiées, les récompenses affichées ne sont pas celles utilisées pendant les tests. Toutes les récompenses positives (les tâches que le robot essaie d'accomplir) sont données avec une récompense de 1, celles qui sont à éviter avec une récompense de -1.

Pour la motivation **Assemblage**, nous constituons trois états motivationnels, représentant les trois étapes de l'assemblage. Depuis la première étape, le robot doit assembler deux pièces pour créer un objet *cadre&moteur* avec succès. Dans la deuxième étape, deux *rmt* sont disponibles, la première est une tâche commandant au robot d'assembler un ventilateur complet, le seconde correspond au fait de briser l'objet *cadre&moteur* alors de la conception du ventilateur (cette seconde *rmt* possède une récompense négative et ramène à la première étape). La dernière étape consiste à déposer le ventilateur sur le tapis roulant, où il est emporté hors de l'atelier. La Figure 6.2 illustre la motivation définie.

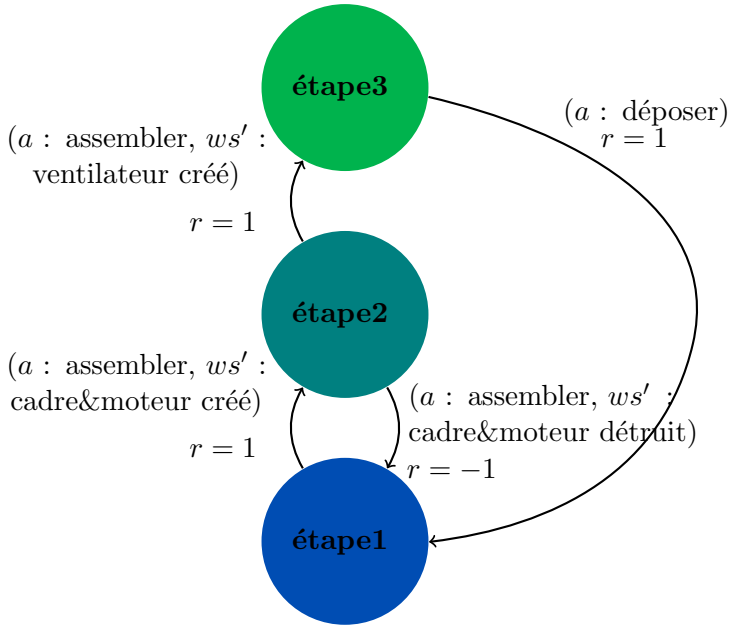


Figure 6.2: Schéma de la motivation **Assembler**.

La motivation **Survie** est constituée de quatre états correspondant aux quatre intervalles d'énergie de la description de l'objectif, comme montré dans la Figure 6.3. Les intervalles sont : $[0-10\%]$ pour l'état *critique*, $[10-40\%]$ pour l'état *faible*, $[40-70\%]$ pour l'état *bon*, et enfin $[70-100\%]$ l'état motivationnel *complet*. L'état passe d'un état à l'autre lorsque la valeur de la ressource *énergie* change, grâce à des *rmt*. Celles-ci donnent une récompense positive lorsque la motivation passe dans à une intervalle où le niveau de la batterie est meilleur, et négative dans le cas contraire. Dans les trois premiers états, une *rmt* permet de rendre disponible une tâche permettant de se recharger une fois. Pour recharger sa batterie, le robot peut donc effectuer cette tâche jusqu'à arriver dans l'état *complet*. Pour finir, une *rmr* est présente dans l'état *critique* et qui inflige une récompense

négative régulièrement, incitant d'autant plus le robot à sortir de cet état.

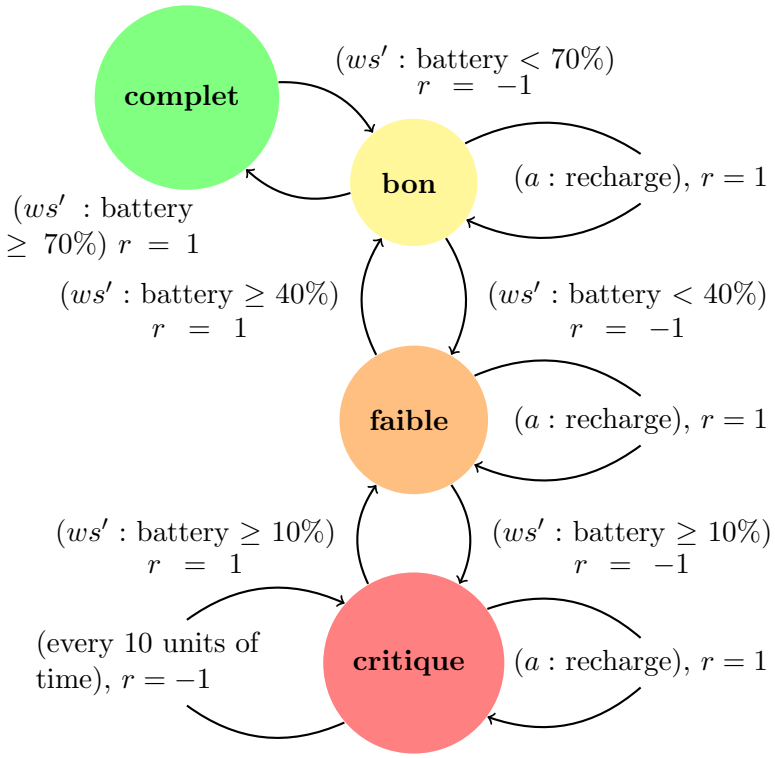


Figure 6.3: Schéma de la motivation **Survie**.

Pour garantir que les chargeurs sont activés et prêts à l'emploi chaque fois que le robot en aura besoin, la motivation **Maintenance** doit réparer les bornes de chargement chaque fois qu'ils présentent un problème. Cette motivation possède deux états, *en marche* et *en panne*, correspondant respectivement à l'état où les chargeurs sont *activés* et à l'état où les chargeurs ne sont pas en état de fonctionner, visibles sur la Figure 6.4. La motivation passe de *en marche* à *en panne* grâce à une *rmt* qui est activée chaque fois que les chargeurs ne sont plus activés, suite à un rechargement raté. Ensuite, une *rmt* est disponible depuis l'état *en panne* et récompense le robot lorsqu'il rétabli l'état des chargeurs.

Pour finir, l'objectif **Zone** est également modélisé à l'aide d'une motivation à deux états. Contrairement à la motivation précédente, sont état n'est pas lié à celui du monde, mais change périodiquement en fonction de l'évolution de la ressource de temps. Les deux états motivationnels, *libre* et *restreinte*, définissent l'état dans lequel la zone est libre d'accès et lorsqu'elle doit être évitée. Les *rewarded motivation transitions* permettant de passer d'un état à l'autre sont des comptes à rebours. Le robot obtient une récompense nulle pour leur réalisation, car ce changement d'état ne peut pas être considéré comme positif ou négatif. Une troisième *rmt* est disponible lorsque la motivation est dans l'état *restreinte*, elle pénalise le robot chaque fois qu'il pénètre dans la zone restreinte. La Figure 6.5 illustre l'automate choisit pour représenter la motivation.

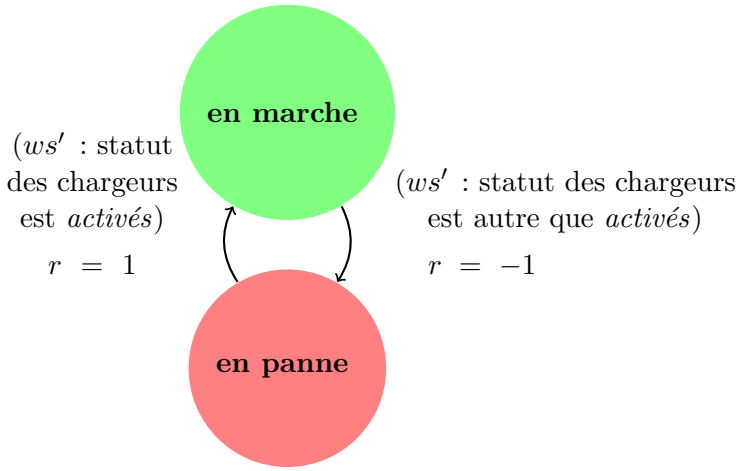


Figure 6.4: Schéma de la motivation **Maintenance**.

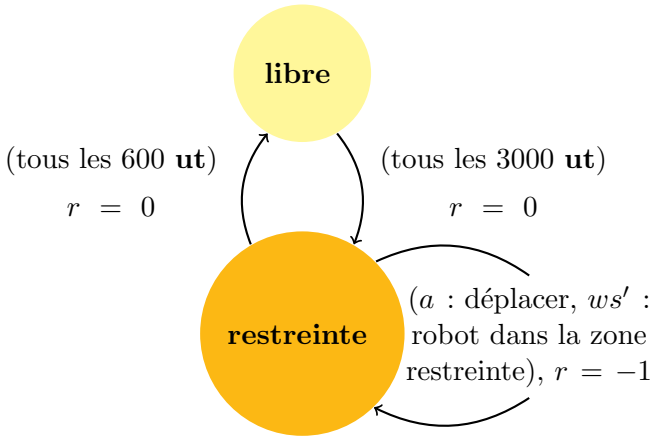


Figure 6.5: Schéma de la motivation **Zone**.

Taille du problème

Dans l'implémentation que nous avons utilisée, le modèle opérationnel du problème présenté a environ 2.4×10^7 états du monde (sans compter la valuation des variables) et 24 actions différentes. Il y a 48 *msv* possibles (c'est-à-dire 48 combinaisons d'état motivationnels).

La taille brute du modèle opérationnel rend compliqué le calcul de politiques, car les algorithmes comme Value Iteration demandent de conserver la fonction de transition du modèle en mémoire. Nous rappelons que la taille de la fonction de transition est $|S|^2 \times |A|$, la taille de l'espace d'état au carré multiplié par la taille de l'espace d'action. Sur la machine que nous avons utilisé et avec les techniques basiques (l'algorithme Value Iteration), l'espace de la mémoire RAM (32 GB) a été complètement saturé sans pouvoir charger la fonction de transition. Si nous souhaitons résoudre le problème complet de façon naïve et en ignorant les ressources, la taille du MDP à résoudre serait de 10^9, plus

difficile encore à calculer.

Cependant, le système que nous avons mis en place permet de calculer des politiques permettant de déclencher des *rmt* pendant l'initialisation. Chaque politique utilise un sous-modèle adapté à la *rmt*, ce qui permet d'utiliser Value Iteration avec des modèles dont l'espace d'état maximum est de l'ordre de 10^5. Le processus de délibération permet dans un second temps d'organiser l'exécution des politiques calculées grâce à la conception d'un *agenda de macro-actions*.

6.2 Implémentation et matériel utilisé.

Pour réaliser ces expérimentations, nous avons utilisé le modèle du problème décrit plus tôt dans ce chapitre ainsi que l'architecture proposée dans les chapitres précédents, écrite en langage Python.

Implémentation. Si l'architecture présentée peut paraître simple, son implémentation requiert un travail assez long et plusieurs milliers de lignes de code. Il y a, en effet, de nombreuses fonctionnalités à écrire, comme la génération des sous-modèles, le calcul des politiques et des fonctions prédisant leurs comportements, ou la génération des effets d'une macro-action lors de la délibération. Et si certaines de ces fonctionnalités, comme le calcul d'une politique via Value Iteration ou la génération d'un modèle MDP à partir d'un sous-modèle et d'une *rmt*, sont assez connues ou simples à implémenter, d'autres opération, comme la résolution des systèmes d'équations, peuvent demander une grande quantité de travail.

En effet, le système proposé étant destiné à résoudre des problèmes de grande taille, le soin apporté à l'optimisation des calculs est très important. Les trois processus les plus coûteux du système sont :

- **Le calcul des $\pi_{\boldsymbol{rmt}}$** : utilisant l'algorithme Value Iteration, ce processus est très coûteux en mémoire vive. Si la hiérarchisation et l'utilisation de sous-modèles permettent d'en limiter le coût, le choix des structures de données s'avère important pour les performances, en diminuant l'espace pour stocker le modèle MDP ou pour y accéder.

- **Les calculs des fonctions $Pr(\boldsymbol{rwt}|ws', \pi)$ et $C_r(rwt, ws, \pi)$** : il s'agit de résoudre des système d'équations de grande taille. Si des méthodes génériques existent, nous leur avons préféré une méthode de résolution personnalisée et optimisée en exploitant les spécificités des systèmes traités. Nous expliquons les biais que nous avons trouvé et comment nous les avons exploités ci-dessous.

- **La création d'un arbre des macro-actions** : calcul récurrent, exécuté plusieurs dizaines ou centaines de fois lors de chaque expérimentation. Il s'agit ici de développer un arbre, partant de la situation présente, et présentant l'ensemble des situations futures en fonction des macro-actions exécutées. Pour l'implémentation de ce calcul, nous avons porté notre effort sur les critères de développement de l'arbre. En effet, chaque nœud ayant en moyenne une quinzaine de fils possible, plus l'exploration en profondeur avance, plus le nombre de nœuds fils à générer est important (15 pour le premier niveau, 225 pour le second, 3 375 pour le troisième, puis 50 000, 760 000, etc.). L'objectif était donc d'essayer de ne développer que les nœuds "intéressants" en proposant plusieurs facteurs limitant le développement de l'arbre.

Nous avons abordé cette problématique Section 5.3.2 et nous vérifions l'efficacité des facteurs sélectionnés dans les configurations testée ci-après (Sections 6.3.2 et 6.3.3).

La réalisation des calculs des fonctions $Pr(rwt|ws', \pi)$ et $C_r(rwt, ws, \pi)$ est principalement basé sur la résolution de grands systèmes d'équations linéaires, dont la complexité temporelle est généralement de $O(N^3)$, avec N le nombre d'état ws possibles. Cependant, nous avons testé les méthodes proposées par les librairies de calcul à notre disposition (utilisant les méthodes de Cholesky, la décomposition LU, etc.). Nous nous sommes aperçus que l'utilisation des outils permettant la résolution de ces systèmes prenaient un temps important pour résoudre les systèmes lorsque le problème n'était plus trivial (à partir de 1000 variables).

Pour parer à ce problème, nous avons choisi de développer notre méthode de résolution, spécialement conçue pour notre problème. En constatant que les systèmes d'équations linéaires étaient généralement creuses (dû au nombre limité de possible états futurs à partir d'un couple (ws, a)), nous avons choisi d'utiliser l'élimination de Gauss-Jordan. En effet, en guidant cet algorithme grâce à nos connaissances sur le système, les transitions buts en l'occurrence correspondant à rwt, nous avons pu accélérer la résolution des systèmes d'équations. Nous avons ensuite vus qu'il était possible d'utiliser la stratégie diviser pour régner afin d'accélérer encore le processus. Cela est dû au fait que les chaînes de markov représentant les politiques π scindent assez souvent l'espace en plusieurs sous-espaces non-connectés. En effet, à partir d'un état ws, il est très fréquent que la politique ne permette d'atteindre qu'une fraction des transitions buts définies. Nous avons donc pu résoudre chaque sous-espace indépendamment, limitant ainsi la taille maximale des systèmes d'équations et les temps de calculs.

Avec l'utilisation de ces méthodes de résolution spécifiques aux systèmes rencontrés, nous avons pu limiter le temps de résolution des systèmes le plus imposant, de plusieurs dizaines de minutes à plusieurs dizaines de secondes (jusqu'à 2 minutes pour les plus longs).

Limitations de l'implémentation. Cependant, certains travaux ayant étés développés tardivement ne sont pas totalement intégrés dans l'architecture utilisée. Les calculs utilisés pour déterminer les distributions des coûts en ressources des politiques sont par exemple moins aboutis que ceux présentés dans ce travail. De plus, le système actuel n'est pas parallélisé et ne fonctionne donc que sur un seul processeur. Notons qu'à priori, tous les processus, mis à part l'exécution de l'agenda, devraient pouvoir être parallélisés.

Matériel. La configuration matérielle utilisée est un ordinateur de bureau multi-processeurs de 3.4 GHz et une capacité de mémoire RAM de 32 GB.

6.3 Tests

Puisque, à notre connaissance, aucun système de décision ne traite en intégralité de problème avec les caractéristiques que nous avons choisies, nous avons décidé d'observer l'efficacité de notre système en mesurant la performance de notre système délibératif et avec plusieurs configurations possibles.

Définition d'une expérimentation. Une expérimentation consiste à exécuter un certain nombre de fois une configuration de notre architecture avec toujours avec le même problème et la même état de départ. Les propriétés d'un expérimentation sont :

- **20 tests** : nous exécutons chaque configuration 20 fois afin de générer un minimum de donnée statistique sur la performance de la configuration.
- **Conditions d'arrêt** : chaque test simule une durée de 21 600 **ut** et s'arrête donc lorsque la valeurs de la ressource de temps atteint cette valeur. De plus, le test s'arrête dès que la ressource de l'énergie passe à 0, puisqu'il décrit une situation ou la batterie est vide et où le robot ne peut donc plus continuer à fonctionner.

Déroulement d'une expérimentation. Lors d'un test, nous pouvons distinguer trois différentes phases de calculs :

- **Initialisation** : il s'agit de la phase la plus complexe puisqu'elle regroupe une grande variété de calculs. Lors de cette phase, le module délibératif détermine l'ensemble des *RWT* dont le système aura besoin, définissant chacun une *rwt* à exécuter plus un ensemble d'autre *rwt* à éviter Ensuite, le module opérationnel calcule pour chaque *RWT* la politique qui correspond ainsi que le comportement de cette politique, les états de fins et leur probabilités, ainsi que leurs coûts en ressources. Pour finir, le module délibératif conçoit des macro-actions à partir des calculs effectués précédemment. Il est alors prêt à effectuer des délibérations.
- **Délibération** : cette phase consiste à générer un arbre des macro-actions, puis à l'élaguer pour obtenir un *agenda de macro-actions*.
- **Exécution** : lors de cette phase, un *agenda de macro-actions* précédemment calculé est exécuté. Au cours de l'exécution, le superviseur vérifie que l'état courant et la valeur des ressources correspond bien à la progression prédite dans l'agenda.

La phase d'initialisation est fixe et est toujours la première à être exécutée. Ensuite, le système alterne entre les phases de délibération et d'exécution comme illustré Figure 6.6.

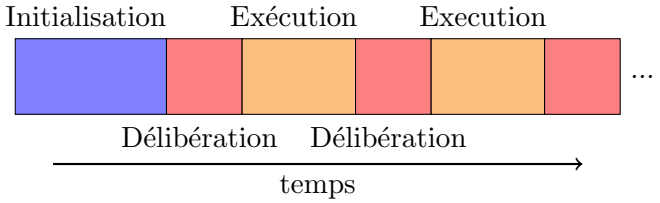


Figure 6.6: Déroulement de l'exécution de l'architecture.
Après la phase d'initialisation, l'algorithme alterne entre des phases de délibération et d'exécution. Les couleurs correspondent aux couleurs des modules en majorité, et utilisés dans la Figure 5.15.

Phase d'initialisation. Quelque soit la configuration testée, la phase d'initialisation est toujours identique. Pour cette raison, nous ne la prenons pas en compte lors des tests de performance des différentes configurations. De plus, puisque les calculs sont toujours les mêmes, nous avons enregistré les résultats produits pour éviter de les calculer à chaque fois et accélérer la vitesse des expérimentations.

Cette phase a donc une durée d'environ 220 secondes et utilise moins de 1 GB de mémoire RAM. Tous les calculs s'effectuent sur un unique processeur. Nous pouvons constater que l'utilisation de méthodes factorisées permet de réduire très considérablement la

charge de calcul : nous avons testé plus tôt le même calcul sans factorisation et la mémoire RAM de 32 GB a été insuffisante pour exécuter le moindre calcul. Les performances de cette phase de calcul sont donc bonnes, en particulier sa consommation de mémoire vive.

Agenda exemple. Nous présentons pour finir un extrait d'un agenda calculé pour le problème présenté, illustré Figure 6.7. Pour des raisons pratiques, les véritables agendas sont très volumineux, nous en avons sélectionné un extrait et l'avons simplifié. Dans la situation décrite, le robot a un objet *cadre&moteur* dans une main et s'apprête à aller chercher des *pales* pour assembler un *ventilateur*. Lors de cette macro-action, selon le temps pris pour aller chercher l'objet, il passe ou non sous le seuil de 40% en énergie. S'il est assez rapide, il enchaîne avec l'assemblage du ventilateur. Dans le cas contraire, il choisit d'aller recharger ses batteries à la borne la plus proche.

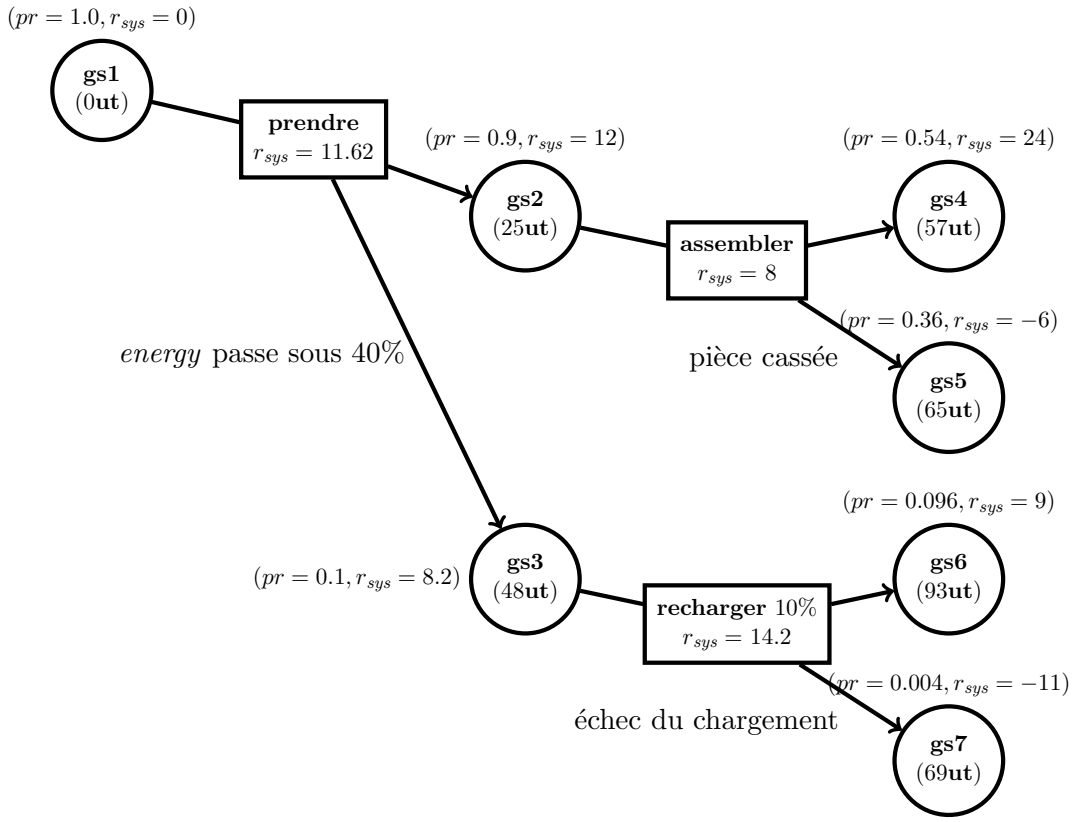


Figure 6.7: Exemple d'agenda de macro-action utilisé par le système.
Les agendas générés par le système délibératif ont régulièrement plusieurs centaines d'états, il s'agit ici d'un extrait d'agenda simplifié.

6.3.1 Configuration nominale

Nous avons choisi comme configuration nominative un réglage des paramètres permettant d'obtenir une récompense accumulée R_{sys} proche du maximum constatée et en conservant un temps de calcul des phases de délibération le plus faible possible. Ces réglages ont été constitués à la main et sont dépendants du problème spécifique que nous traitons.

L'impact des paramètres et l'influence du problèmes sur le choix de leur configuration sera discuté à la fin de ce chapitre.

Une configuration est composée de quatre paramètres, dont les trois premiers sont définis Section 5.3.2. La configuration nominative choisie est :

- **Horizon** : 180 **ut**. Il s'agit du facteur de limitation du développement de l'arbre des macro-actions. Celui-ci détermine à partir de quel coût en ressource de temps le développement de l'arbre s'arrête.

- **Probabilité minimale** : 0.005. La probabilité minimale d'un nœud pour pouvoir le développer. Ce paramètre permet d'éviter de passer trop de temps à développer les branches de l'arbre trop peu probables.

- **Profondeur maximale** : 5. Le dernier facteur limitant concerne la profondeur maximale de l'arbre des macro-actions. Ce facteur permet d'éviter d'enchaîner un nombre très important de tâches courtes et ayant un forte taux de succès. Ces dernières font en effet exploser le développement de l'arbre et, à notre avis, ont une probabilité limitée d'être intéressantes étant donné leur coût en calculs.

- **Exécution de l'agenda** : complète. Il s'agit de la méthode d'exécution de l'agenda. Ici, l'agenda sera exécuté jusqu'au bout tant que l'agenda est valide. Nous verrons dans les dernières configurations testées d'autres méthodes d'exécution et leurs propriétés.

Situation initiale. Nous avons choisi pour tester l'efficacité de notre système de délibération de commencer les expérimentations dans la situation suivante :

- le robot commence près du tapis roulant et avec les freins désactivés,

- les deux portes sont ouvertes,

- aucun des cinq objets ne sont disponibles, et les stockages ainsi que les mains du robots sont vide,

- les chargeurs fonctionnent normalement,

- la batterie commence chargée à 55% et la ressource du temps commence à 0 **ut**.

Les états des motivations au départ sont :

- l'objectif **Assembler** est dans la première étape, où l'objet *cadre&moteur* doit être assemblée,

- la motivation **Survie** commence dans l'état *bon*,

- la motivation **Maintenance** est dans l'état *en marche*, puisque les chargeurs sont activés,

- l'objectif **Zone** commence restreint pour 600 unités de temps. Tout au long de l'expérimentation, la zone est restreinte 600 **ut** tous les 3 600 **ut**, soit six fois 600 **ut**.

Évaluation de l'efficacité de l'architecture. Pour évaluer les performances du système, nous nous intéressons à deux facteurs :

- **Récompense accumulée** : la récompense R_{sys} accumulée par le système en exécutant les rmt des motivations.
- **Temps de délibération** : le temps réel mis par le système pour effectuer les délibérations tout au long des expériences.

Résultats

Nous étudions à présent le détail des tests effectués avec la configuration nominative et ses performances.

Description du déroulement d'un test. Pour donner une idée de ce que produit l'exécution de l'architecture, nous décrivons de façon globale comment un test utilisant la configuration nominale se passe.

Tout d'abord, la majorité du temps, le robot effectue sa mission d'assemblage, ne l'interrompant que pour récupérer de l'énergie. Lorsqu'il assemble des ventilateurs, le robot utilise toujours le bon poste de fabrication lorsque la zone est libre, et le mauvais poste lorsque celle-ci est restreinte. Il arrive que le robot pénètre par accident dans la zone restreinte, le plus souvent à cause de l'aspect aléatoire des déplacements, mais il arrive rarement qu'il y pénètre à cause d'une mauvaise estimation des coûts en ressources. Le robot prend trop de temps pour exécuter les tâches précédentes et arrive dans la zone restreinte alors qu'il avait prévu qu'elle sera encore libre.

La plupart du temps, le robot conserve son niveau d'énergie au-dessus de 70% et ne descend que rarement sous les 40%. Il varie les rechargements à 10% et 25%. De plus, lorsque que les chargeurs dysfonctionnent, le robot s'occupe de les remettre en état de fonctionner rapidement.

Statistiques. Pendant les expérimentations, nous avons enregistré des informations relatives à la ressource de temps investie pour chaque motivation, ou à la structure de l'arbre des macro-actions généré. Avec cette configuration, le système a effectué une moyenne de 108 cycles délibération / exécution avant d'atteindre la limite des 21 600 unités de temps. En moyenne, les arbres des macro-actions contenaient 7 000 nœuds, pour une profondeur de 4.6. Toujours en moyenne, chaque nœud avait 5.64 macro-actions qu'il était possible d'exécuter, et chaque macro-action avait 2.73 fils possibles.

Tout au long des tests, nous avons enregistré le coût réel en ressource de temps utilisé pour chaque macro-action. Le tableau suivant donne la ressource de temps dépensé pour chaque motivation (la motivation **Zone** n'est pas présente car elle ne propose aucune macro-action).

motivation	temps (ut)
Assemblage	82%
Survie	16%
Maintenance	2%
total	100%

Performance. L'efficacité du système avec la configuration choisie est mesurée à la récompense R_{sys} accumulée en moyenne ainsi qu'à la somme des durées des délibérations pendant les expérimentations. Pour la configuration nominale et en moyenne, le système à accumulé une récompense de 4 440, et a passé 257 secondes réelles à délibérer (avec notre configuration matérielle) tout au long des 108 phases de délibération.

Pour finir, nous donnons dans la Figure 6.8 l'évolution moyenne de la récompense accumulée R_{sys} pendant les tests, ainsi qu'une moyenne glissante des temps de délibération.

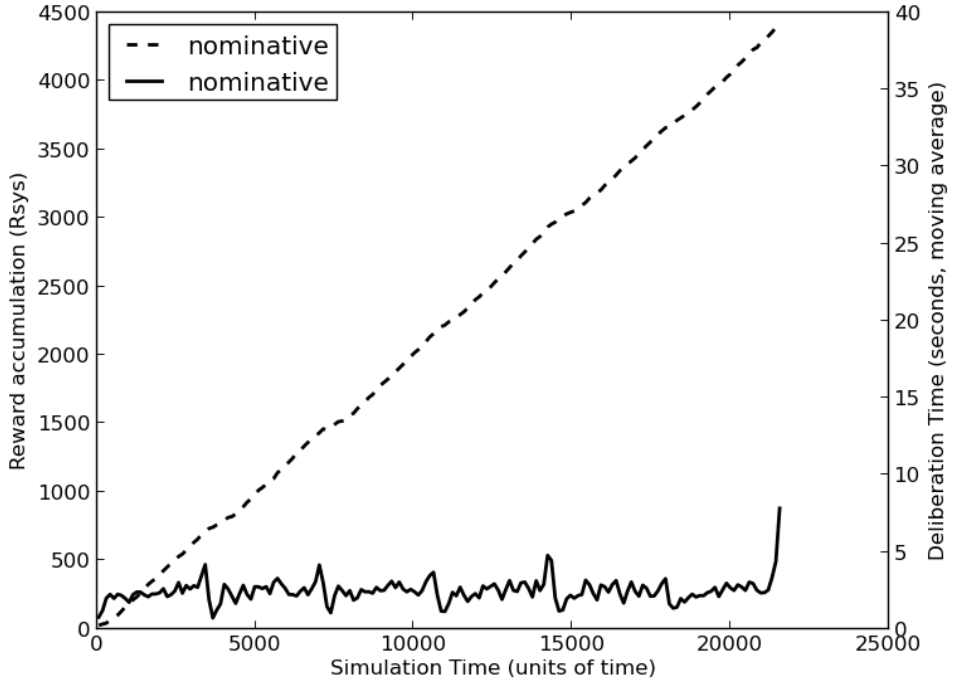


Figure 6.8: Performances de la configuration nominale.

6.3.2 Limite de l'horizon

Pour la seconde configuration testée, nous allons faire varier l'horizon utilisé pour le développement de l'arbre des macro-actions. Nous allons comparer trois configurations :

- **Horizon** : 180 **ut**. Il s'agit du facteur de limitation du développement nominal.
- **Horizon** : 100 **ut**. Avec ce paramètre, les délibérations donneront des agendas dont la longueur des scénarios est plus courte. Il y aura donc un nombre de cycles délibération / exécution plus important.
- **Horizon** : 1 **ut**. Ici, le but est de limiter le développement de l'arbre des macro-actions à son maximum. De cette manière, l'arbre de développe que le nœud de départ. Ainsi, avec cette limitation, l'agenda retourné choisira uniquement la première macro-action ayant la plus forte espérance de récompense. Ce test permettra de constater l'importance de l'ordonnancement des macro-actions dans le temps.

La Figure 6.9 permet de comparer les performances des trois configurations entre elles.

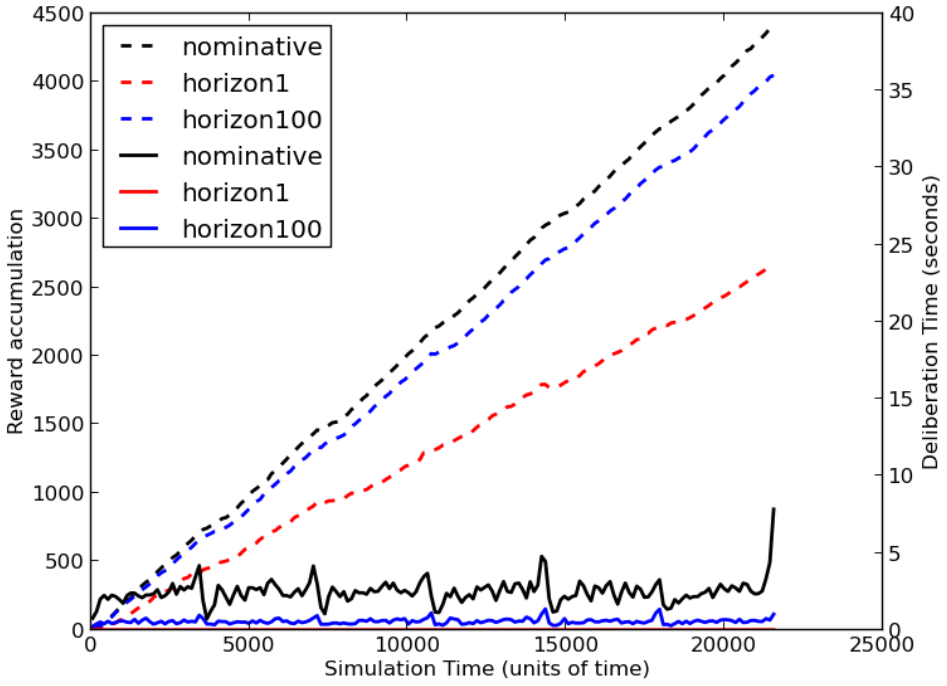


Figure 6.9: Performances des configurations *nominale*, *Horizon100* et *Horizon1*.

Le tableau suivant compile les données recueillies sur l'exécution des trois configurations.

donnée	*nominale*	*Horizon100*	*Horizon1*
R_{sys}	4 440	4 060	2 650
nombre de cycles	108	157	430
temps total délibération (s)	257	86	1
nœuds développés	7 000	1 800	7

Analyse. Nous pouvons commencer par analyser les résultats pour la configuration avec *Horizon1*. Nous pouvons constater que ses performances en matière de récompense accumulée sont très inférieures aux deux autres configurations envisagées, mais que son coût est évidemment très bas. Cela nous permet d'affirmer que la capacité à **enchaîner plusieurs macro-actions** est importante et que la récompense promise pour une seule macro-action n'est pas suffisante pour déterminer la macro-action à exécuter. De plus, cela souligne l'importance de la **capacité à prévoir les effets des macro-actions** de façon précise.

Ensuite, la comparaison entre les performances de plusieurs configurations ayant des horizons différents permet de faire la constatation assez évidente que plus l'horizon est éloigné, plus la récompense accumulée se rapproche de l'optimum et plus le coût des délibérations augmentent. De plus, limiter l'horizon de l'arbre fait varier le nombre de

délibération et donc d'agenda de macro-action nécessaire pour arriver au bout des 21 600 **ut** de l'expérience. En revanche, si le nombre de délibération augmente, le coût pour chaque délibération diminue plus encore. Ainsi, le temps total passé à délibérer est inférieur à la configuration nominale.

6.3.3 Autres facteurs de limitation du développement de l'arbre des macro-actions

Pour la seconde configuration testée, nous allons faire varier les autres facteurs utilisés pour limiter le développement de l'arbre des macro-actions. Il s'agit des facteurs de probabilité minimale pour développer un nœud et de la profondeur maximale de l'arbre. Ces paramètres permettent respectivement d'éviter de prendre du temps pour les scénarios les plus improbables et d'éviter les scénarios avec un très grand nombre de macro-actions courtes, jugés peu intéressant pour leurs coûts. Nous allons comparer trois configurations :

- **Nominale** : la probabilité pour développer un nœud est que sa profondeur doit être inférieur à 5, et la probabilité d'arriver à ce nœud doit être supérieur à 0.005.
- **LimiteForte** : ici, les limitations sont plus importantes, avec une probabilité minimale de 0.02 et une profondeur de 4.
- **LimiteFaible** : au contraire, cette configuration assouplit les conditions pour développer un nœud. La profondeur maximale devient 7 et le facteur de probabilité minimal devient 0.001. L'arbre des macro-actions sera donc plus étoffé.

La Figure 6.10 permet de comparer les performances des trois configurations entre elles.

Le tableau suivant compile les données recueillies sur l'exécution des trois configurations.

donnée	*nominale*	*LimiteForte*	*LimiteFaible*
R_{sys}	4 440	4 130	4 470
nombre de cycles	108	116	104
temps total délibération (s)	257	72	2 201
nœuds développés	7 000	1 800	52 700

Analyse. Nous pouvons commencer par observer les performances de la configuration *LimiteFaible*. Nous pouvons remarquer que, pour le même horizon (180 **ut**), le nombre de nœuds développés et plus important grâce aux limitations plus faibles sur le développement de l'arbre des macro-actions. Cependant, le temps total passé à délibérer est plus important pour un gain négligeable. Nous pouvons d'ailleurs remarquer que le nombre des délibérations et donc des agendas utilisés n'a que peu diminué, ce qui montre l'intérêt limité des scénarios que nous avons choisi d'ignorer.

Dans le cas de la configuration *LimiteForte*, Nous pouvons constater qu'en échange d'une baisse raisonnable de la récompense accumulée, il est possible de diminuer de façon significative la durée moyenne de la phase de délibération. En revanche, nous pouvons voir que le nombre de délibérations à augmenté, signe que le nombre des scénarios qui n'ont pas étés calculés à cause des limitations a été plus important.

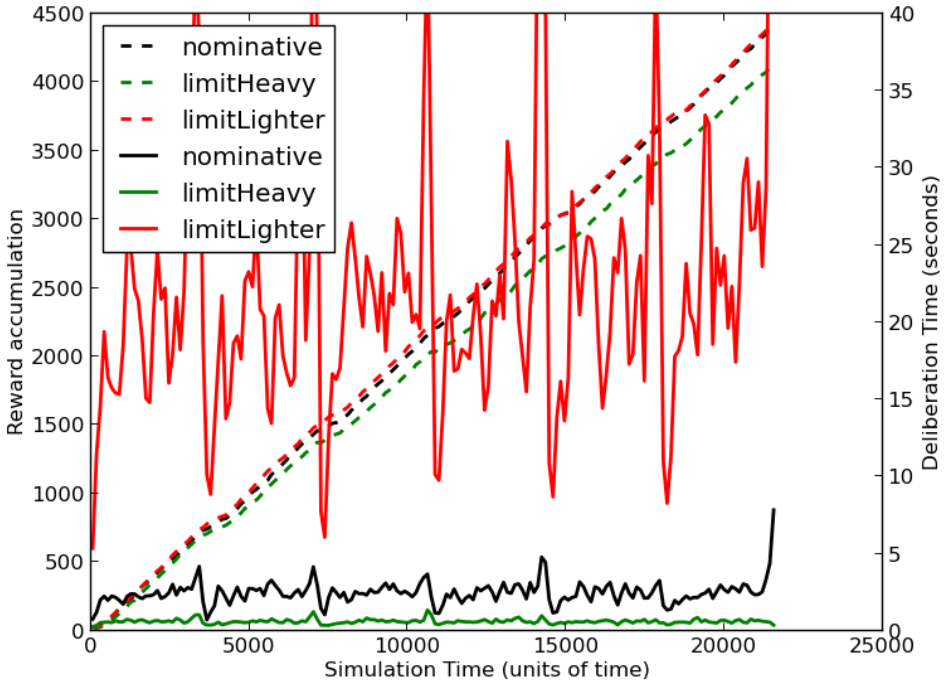


Figure 6.10: Performances des configurations *nominale*, *LimiteForte* et *LimiteFaible*.

6.3.4 Exécution limitée de l'agenda

Pour le troisième test, nous avons choisi d'utiliser la configuration nominale, mais en faisant varier la manière d'exécuter l'agenda. Dans tous les tests précédents, les agendas étaient parcourus jusqu'à leur fin ou lorsque l'état global à la fin de l'exécution d'une macro-action ne se trouvait pas dans l'arbre (phénomène dû aux approximations de la prédiction des consommations en ressources des macro-actions). Nous souhaitons à présent tester si, en stoppant volontairement l'exécution de l'agenda de façon prématurée, il est possible d'augmenter les performances du système. Notre intuition est que plus nous nous approchons des feuilles de l'arbre, moins les décisions sont proches de l'optimal. Cela pourrait être dû au fait que les choix effectués à la racine de l'agenda dépendent des calculs fait pour les nœuds plus profond (rappelons que les calculs partent des feuilles et remontent jusqu'à la racine). Nous allons donc comparer trois configurations :

- **Nominale** : il s'agit là de la méthode d'exécution normale des agendas.
- **StoppeDeuxTiers** : ici, une fois que la ressource de temps consommée depuis le début de l'exécution de l'agenda dépasse les deux tiers de l'horizon (donc 120 **ut**), à la fin de la macro-action en cours d'exécution, l'exécution de l'agenda s'arrête. Un nouveau cycle délibération / exécution commencera donc.
- **StoppeUnTiers** : idem que pour la configuration *StoppeDeuxTiers*, mais se stoppe à seulement un tiers de l'agenda, donc à partir de 60 **ut**.

Les performances des trois configurations sont comparées Figure 6.11.

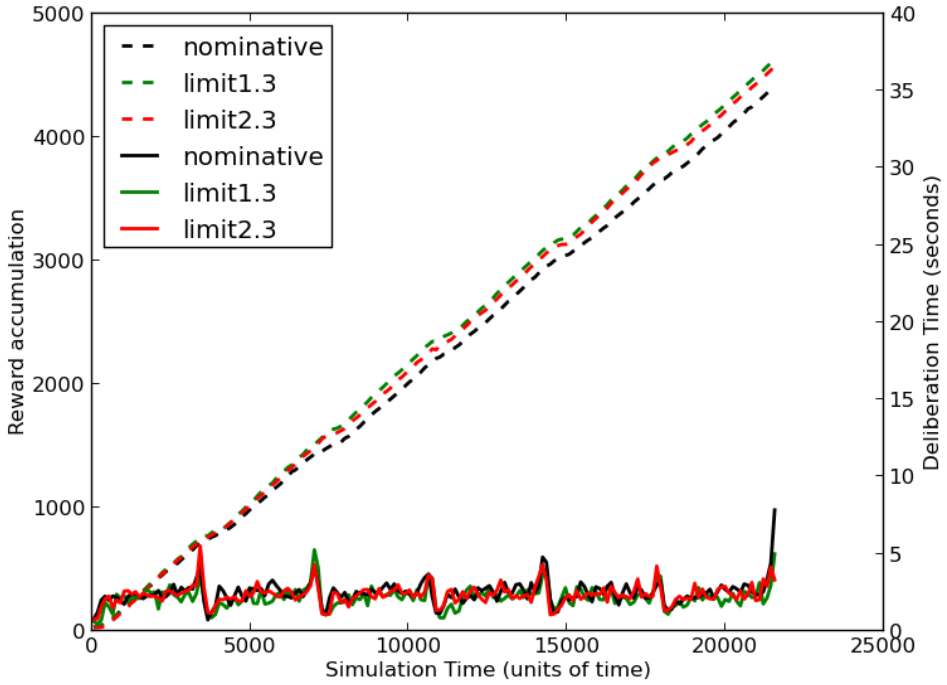


Figure 6.11: Performance des configurations limitant l'exécution de l'agenda de macro-actions.

Le tableau suivant compile les données recueillies sur l'exécution des trois configurations.

donnée	*nominale*	*StoppeDeuxTiers*	*StoppeUnTiers*
R_{sys}	4 440	4 580	4 630
nombre de cycles	108	146	236
temps total délibération (s)	257	340	486
nœuds développés	7 000	7 000	7 000

Analyse. Nous pouvons voir sur les performances que plus l'agenda est stoppé tôt, plus la récompense accumulée augmente. Cela confirme notre hypothèse selon laquelle les décisions du début de l'agenda sont plus proches des décisions optimales que les décisions proches de la fin de l'agenda.

La configuration *StoppeUnTiers* est la configuration avec laquelle nous avons réussi à obtenir le plus de récompense parmi tous les tests que nous avons effectués. Cependant, nous pouvons constater que le gain par rapport à une exécution normale de l'agenda est faible. De plus, si stopper prématurément les agendas ne change pas le temps pour en calculer un (les courbes se confondent sur la figure), cela augmente le nombre de délibération nécessaire et donc le temps total passé à délibérer.

6.4 Discussion

L'architecture proposée a été implémentée et testée en utilisant une machine de bureau, sur un seul processeur de 3.4 GHz et en utilisant toujours moins de 2 GB de mémoire vive. Nous avons proposé de résoudre un problème de taille non-triviale et impossible à résoudre avec les méthodes classiques de résolution complète. Le problème posé propose un modèle probabiliste dans lequel de multiples objectifs concurrents doivent être satisfaits. Ces objectifs sont structurés sous la forme de *motivations* et n'ont pas de fin, permettant d'exprimer des tâches répétitives ou des besoins constants. Le problème n'a pas de fin explicite et utilise des ressources (pour modéliser le niveau de charge de la batterie ainsi que le temps). À notre connaissance, aucun travail n'a choisi de traiter de problème ayant l'ensemble de ces caractéristiques.

Le problème exemple traite quatre motivations et permet de décrire une situation d'autonomie en milieu fermé satisfaisante, avec des objectifs contradictoires. Le problème posé à l'agent est complexe, il doit choisir comment répartir son temps entre les objectifs posés. L'architecture proposé permet au robot de considérer l'ensemble des conséquences des macro-actions qu'il peut exécuter : les différentes situations finales possibles, leurs effets sur les objectifs et leurs utilisation en termes de ressources. Avec le système de délibération, nous avons réduit un problème ayant de multiples objectifs structurés à un problème de sélection parmi un ensemble restreint de macro-actions. Nous pouvons en déduire que ce phénomène résulte bien d'une forte abstraction du problème et permet au robot de considérer uniquement l'ensemble des grandes orientations qui sont à sa disposition à un moment donné.

Le modèle du monde proposé est composé d'environ 2.4×10^7 états, sans compter les ressources. Sur les machines testées, générer et garder en mémoire la fonction de récompense de ce modèle s'est avéré impossible, car consommant toute la mémoire vive disponible. En effet, la fonction de transition est une matrice de taille $|S|^2 \times |A|$. Lorsque la taille du problème devient important, la mémoire nécessaire pour stocker la fonction de transition explose.

L'utilisation de l'architecture présentée nous a permis au contraire de nous attaquer à ce problème et avec une utilisation modeste des ressources informatiques. En hiérarchisant la résolution en deux processus (formation des macro-actions et délibération), nous avons pu apporter une solution cohérente au problème, en calculant des *agendas de macro-actions*.

Lors des tests réalisés, nous avons pu vérifier que les choix faits pour résoudre le problème créent un comportement cohérent avec le problème posé. Le robot s'affaire bien à assembler des ventilateurs en évitant la zone lorsqu'elle est restreinte et utilise le meilleur poste de fabrication à sa disposition. Il stoppe sa tâche d'assemblage pour maintenir son niveau d'énergie élevé et ne prend pas le risque d'être pénalisé lorsque le niveau de la batterie est faible. De plus, il remet rapidement en route les chargeurs lorsque ceux-ci subissent un dysfonctionnement.

Nous avons également pu vérifier l'utilité de limiter le développement de l'arbre des macro-actions et l'influence des limitations sur les performances, ces limitations allégeant considérablement la quantité de calcul sans avoir de réel impact sur les performances. Pour finir, nous avons vu que stopper prématurément l'exécution des agendas était une piste permettant d'augmenter l'optimalité de notre solution, celle-ci étant limitée par la recherche locale proposée pour la délibération. Nous avons également exploré la possibilité de rendre le processus de délibération *anytime*. Cependant, si le développement de l'arbre des macro-actions peut être rendu *anytime* (et parallélisé), le processus qui élague l'arbre

pour donner l'agenda de macro-actions ne peut s'effectuer que lorsque l'arbre est fini de développé (car le processus débute par les feuilles de l'arbre). Ainsi, il subsisterait toujours une phase de calcul non-triviale à exécuter après avoir stoppé le développement de l'arbre.

Enfin, plusieurs problèmes posés restent à ce jour non résolus, concernant le modèle utilisé dans ce travail ou ses capacité d'adaptation à un problème changeant.

Tout d'abord, nous n'avons pas trouvé de méthode pour calculer la véritable courbe des coûts en ressources des macro-actions en un temps raisonnable. Nous avons fait le choix d'utiliser une méthode arbitraire, donnant la plupart du temps des résultats utilisables. Ce manque de précision reste cependant gênant et provoque des situations où un agenda calculé devient faux, suite à une trop grande différence entre les coûts réels et estimés. Noter que dans ce cas, le décalage réalité / prédiction est détecté et un nouvel agenda est calculé.

Ensuite, le système subit des limitations inhérentes à la hiérarchisation. Si le problème que nous avons choisi se prête assez bien à la hiérarchisation, d'autres problèmes peuvent échouer à être hiérarchisés. Dans le système proposé, les performances dépendent du fait que les actions agissent sur un nombre restreint des paramètres du problème. Dit autrement, si une l'exécution des actions permettent d'aller dans un très grand nombre d'états, alors le problème sera peu factorisé. Cet aspect est renforcé par l'utilisation de macro-actions : plus les actions utilisées par une macro-action agissent sur un ensemble restreint de variables, plus la macro-action sera facile à calculer.

Pour finir, le système proposé permet que les récompenses obtenues pour la réalisation des *rmt* soient modifiées lors de l'exécution et sans nécessité de calculs supplémentaires. C'est également vrai pour les seuils de ressources des *rmt*. Cependant, si l'action à accomplir pour réaliser une *rmt* change, ou si la structure de l'automate d'une motivation est modifiée, alors la phases d'initialisation doit être recommencée au moins en partie.

Chapitre 7

Conclusion et ouvertures

Contents

Dans ce mémoire de thèse, nous nous sommes intéressés à la résolution d'un problème probabiliste sans limite de temps explicite. Ce problème comportait de multiples objectifs possiblement concurrents ainsi que de plusieurs ressources, en particulier l'énergie et le temps. La base et la clé de la résolution de ce problème était la proposition d'un nouveau modèle pour représenter des objectifs. Ce nouveau modèle, les *motivations*, devait permettre d'exprimer des objectifs complexes, structurés et permanents. Le second objectif était le développement d'une architecture délibérative permettant de traiter les *motivations*. Pour pouvoir résoudre des problèmes complexes et de taille importante et garantir une autonomie dans le choix des objectifs, l'architecture devait proposer des méthodes permettant d'utiliser des méthodes de décision de grande taille en évitant l'explosion en coût des calculs et en garantissant la capacité du système à prédire le résultat de ces actions.

7.1 Contributions

Pour répondre à la problématique, nous avons commencé par une étude de différentes modélisations des objectifs spécifiquement adaptés au domaine de la décision probabiliste : les états buts ainsi que la hiérarchisation de tâches. Nous avons vu que leurs structures impactent fortement les techniques de décision utilisables ainsi que leurs performances. Nous avons ainsi constaté que les états buts ne permettaient pas de constituer des objectifs complexes car ils ne permettent pas d'exprimer des enchaînements sans faire exploser la taille du modèle. La hiérarchisation de tâches permet de découper de façon récurrente les objectifs en sous-objectifs en utilisant des états buts définis sur des sous-domaines, et de répondre en partie à la problématique de calculer des problèmes importants en un temps raisonnable. Cependant, la capacité à construire des objectifs complexes ne nous semble pas acquise et la méthode de hiérarchisation semble très dépendante du problème traité et difficile à maîtriser.

Nous avons donc choisi de proposer une manière nouvelle de décrire les objectifs en nous appuyant sur la structure des machines à nombre d'états fini. En effet, les automates sont des structures simples et claires, dont l'état change selon certaines conditions. Ceux-ci étant capable d'encoder des conditions, des boucles et des embranchements, nous avons montré comment un automate pouvait être utilisé pour décrire l'évolution d'un objectif complexe. Les automates sont composés d'états et de transitions, pour lesquels nous avons trouvé une signification du point du vue d'un objectif. Les états traduisent les différentes étapes ou situations qui font qu'un objectif soit proche ou loin de sa résolution ou qui peut avoir un impact sur les actions à entreprendre. Les transitions quant à elles définissent les conditions de passage d'un état à un autre. Dans le cadre d'un objectif, il s'agit d'états clés ou d'actions à accomplir. Afin de donner une valeur, positive ou négative, de la progression de l'état de la motivation, nous avons proposé de placer une valeur de récompense sur les transitions, obtenues lors de leur déclenchement. Nous les nommons *rewarded motivation transitions* (*rmt*). Ainsi, en construisant un automate composé d'un ensemble d'états pertinents et de transitions reliant ces états de façon logique, nous pouvons décrire l'évolution de l'objectif encodé en déclenchant les transitions lorsque les actions associées sont exécutées, et suivre l'évolution de la récompense obtenue. Nous nommons un objectif décrit avec ce nouveau modèle une **motivation**.

Avec ce nouveau modèle, nous pouvons décrire de multiples objectifs complexes et compatibles avec des scénarios sans limites de temps. De plus, ce modèle a l'avantage de ne pas faire exploser la taille de l'espace d'un MDP. L'ensemble des motivations définies remplace la traditionnelle fonction de récompense. Cela nous permet alors de

parler de deux modèles distincts : un modèle intentionnel, les motivations, et un modèle opérationnel, qui représente uniquement le robot, son environnement et les interactions qu'il peut effectuer.

Une fois le nouveau modèle des objectifs défini, nous avons développé un processus de planification permettant de calculer une solution pour une situation donnée. Nous avons choisi de définir comme critère à optimiser la somme des récompenses obtenues par l'ensemble des motivations et divisée par la ressource de temps dépensée. Il s'agit en fait du critère utilisé par les MDP classiques, adapté à notre problème.

Avant de passer à la planification, nous avons dû choisir la manière de représenter les ressources au sein de notre modèle. En effet, les ressources sont par nature représentées par des valeurs numériques, mais de nombreux travaux choisissent de discrétiser leurs niveaux afin de fonctionner avec un modèle uniquement discret et pouvoir ainsi utiliser les méthodes de planification classiques. Cependant, nous avons choisi de ne pas effectuer cette discrétisation à cause de la ressource représentant le temps écoulé. Cette ressource n'étant pas bornée (le temps s'écoule sans fin), discrétiser sa valeur créerait une infinité de plages de valeurs. Nous avons ainsi choisi de conserver une représentation hybride de l'état, avec une partie discrète habituelle, et une partie continue uniquement pour les ressources.

Pour la recherche d'une solution au problème, nous avons dû trouver une méthode pour déclencher les *rmt* des motivations afin d'obtenir la récompense. Nous avons donc choisi de constituer une planification en deux temps. Dans un premier temps, l'objectif était de calculer des politiques utilisant le modèle opérationnel et terminant par la transition définie pour chaque *rmt*. Nous introduisons pour cela le **module opérationnel**, qui effectue tous les calculs relatifs à l'exécution des *rmt*. Pour garantir qu'une politique visant le déclenchement d'une *rmt* n'en déclenche pas une autre (sans être voulue, à cause de la nature probabiliste du modèle), nous avons choisi d'envoyer au module opérationnel, pour chaque combinaison d'états des motivations, la *rmt* visée ainsi que les *rmt* pouvant être déclenchées par erreur. Une fois calculées par le module opérationnel, nous souhaitons créer des macro-actions modélisant l'exécution des politiques. Dans un deuxième temps, nous avons constitué un processus de planification utilisant les macro-actions au sein d'un second module, le **module délibératif**. La présence de ressources nous empêche d'avoir un nombre fini d'états et donc de choisir une macro-action à exécuter pour chacun, nous avons donc mis au point une méthode de planification locale.

Avant de calculer les politiques demandées par le module délibératif, nous avons choisi de définir le modèle du monde utilisé à l'aide d'opérateurs probabilistes STRIPS (des PSO). Pour des problèmes complexes tels que ceux que nous souhaitons résoudre, les robots exécutent des tâches diverses, utilisant chacune un certain nombre de facultés du robot, mais rarement toutes. Nous en avons conclu qu'il était fortement probable qu'une tâche pouvait être définie sur un modèle réduit, contenant un espace d'état et d'action plus petit. Pour profiter de cette particularité, nous avons défini le modèle du monde de manière à pouvoir en extraire des sous-modèles spécifiques aux *rmt* que nous souhaitons atteindre. Un sous-modèle peut alors être utilisé à la fois pour calculer une politique et construire une macro-action, tout en garantissant que les valeurs des variables d'états non-présentes dans le modèle du sous-monde ne seront pas modifiées lors de l'exécution de la politique.

Pour exécuter les *rmt* envoyées par le module délibératif, nous avons modifié un MDP afin de calculer une politique optimale. Pour calculer une politique qui termine sur un

ensemble de transitions données, nous avons mis en place, pour le calcul de la politique uniquement, un état supplémentaire virtuel qui sert de puits et n'ayant aucune action disponible. Avant l'utilisation de l'algorithme Value Iteration, nous redirigeons les transitions terminales vers le puits. De cette façon, nous pouvons garantir que Value Iteration ne diffuse pas de récompense par-delà les transitions correspondant aux rmt.

Une fois les politiques calculées et afin de pouvoir utiliser les politiques calculées comme des actions de plus haut niveau, nous avons cherché une méthode permettant d'obtenir le modèle d'exécution d'une politique stoppante calculée à partir d'une rwt. Nous avons commencé par modifier une méthode existante, utilisant des systèmes d'équations linéaires, afin de déterminer la probabilité de terminer l'exécution de la politique par une rwt donnée et pour chaque état de départ, $Pr(rwt|ws, \pi)$. Pour pouvoir utiliser une politique définie comme une action, il était nécessaire de connaître la quantité de ressources consommées lors de son exécution. Nous n'avons pas trouvé de méthode exacte qui soit utilisable en un temps raisonnable et avons donc opté pour une méthode approximative. En utilisant le résultat du calcul précédent, nous avons pu utiliser un nouveau système d'équations afin d'obtenir une approximation de la consommation en ressource lorsque, depuis un état donné, l'exécution de la politique finissait par une rwt donnée, $C_r(rwt, ws, \pi)$.

À partir des calculs précédents, nous avons pu définir une méthode pour prédire les effets de l'exécution d'une politique, nous permettant de définir nos macro-actions. À partir des effets prédits d'une macro-action sur le modèle du monde (transition finale et consommation en ressources), nous avons pu déduire les effets sur l'état des motivations.

Avec notre modèle des macro-actions fonctionnel, nous sommes donc dans la capacité de connaître, de façon probabiliste, les situations dans lequel se trouvera le robot à la fin d'une macro-action. Nous pouvons donc en déduire la quantité de récompense obtenue ainsi que les tâches à accomplir dans cette situation future. Dans l'impossibilité d'utiliser des méthodes globales (la présence des ressources crée une infinité d'états possibles), nous avons choisi d'utiliser une méthode d'exploration locale. Il s'agit d'une exploration sous la forme d'un arbre de recherche limité par un horizon défini avec la ressource de temps. Pour des raisons de performances, nous avons choisi de contraindre le développement afin de ne conserver que les scénarios les plus probables. Une fois calculé, nous élaguons l'arbre de façon pour donner un **agenda de macro-actions**. celui-ci sélectionne la meilleure macro-action à exécuter pour les différents états qu'il est possible de rencontrer lors de l'exécution. Ainsi, nous déterminons la macro-action permettant de maximiser la récompense obtenue par les motivation et par unité de temps.

Avec la méthode de planification que nous avons présentée, nous avons été capable d'utiliser le modèle des motivations défini plus tôt et de calculer un plan maximisant la récompense obtenue, tout en conservant une forte capacité de prédiction. Nous avons pu définir un processus en deux étapes : la création de macro-actions résolvant une tâche d'une motivation grâce des méthodes de planification complète, suivi d'un calcul de planification local utilisant les macro-actions définies. De plus, nous avons pu déterminer des méthodes exploitant la structure du problème résolu pour diminuer la taille des calculs effectués. Cependant, la méthode locale choisie ne permet pas de résoudre un problème sans limite de temps explicite.

Pour permettre au robot de fonctionner et de sélectionner les macro-actions à exécuter tout au long d'une expérience sans limite de temps, nous avons proposé l'utilisation d'une architecture délibérative. Cette architecture nous permet à la fois de donner la méthode utilisée afin d'exécuter les agendas de macro-actions calculés ainsi

que de définir comment nous pouvons, en alternant exécution et délibération, résoudre le problème dans le temps.

Pour superviser l'exécution des agendas de macro-actions produits, nous avons introduit le **module superviseur**. Lors de l'exécution, ce module reçoit un agenda et s'occupe d'exécuter les politiques correspondant aux macro-actions en envoyant des ordre au système sensori-moteur du robot, qui lui retourne les données des capteurs. Le module superviseur détecte lorsqu'une politique se termine (au moment où une *rwt* est exécutée) et détermine à l'aide de l'agenda la prochaine macro-action à exécuter.

Ce module superviseur permet également de créer un cycle délibération / exécution. Pour cela, il rend également compte au module délibératif l'état de son avancée, ses possibles erreur (à cause de l'approximation des coûts en ressources) et lorsque l'exécution a atteint la fin de l'agenda. Dans les deux derniers cas, l'exécution se met en pause et ne reprend qu'avec un nouvel agenda produit par le module délibératif.

Enfin, nous avons effectué des tests expérimentaux afin de vérifier la validité de l'architecture proposée en simulation. En l'absence de planificateur concurrent pour la problématique proposée, nous avons effectué des expérimentations en utilisant un problème de grande taille, proche de l'exemple récurrent utilisé, présentant un robot manufacturier dans un atelier où celui-ci doit assembler des ventilateurs. Le robot doit également subvenir à ses besoins en énergie, maintenir les bornes qui lui permettent de se recharger en état de fonctionner, ainsi qu'éviter de pénétrer dans une zone qui est périodiquement restreinte. Le problème est représenté à l'aide du modèle de PSO proposé et les objectifs à l'aide de motivations. Le test est réalisé sur une durée importante permettant de tester la capacité de l'architecture à résoudre le problème dans la durée. L'exemple proposé correspond parfaitement à la problématique posée, dispose de plusieurs objectifs, simultanés et concurrents, et propose de nombreuses possibilités d'action au robot. Adopter un comportement logique dans ce problème complexe est pour nous une preuve que le système décisionnel répond à la problématique posée. Pour étayer les tests, nous avons proposé plusieurs configuration de notre architecture et avons analysé les comportements observés et évalué les performances.

Pour la configuration nominale choisie, les résultats ont été convaincant. Le robot adopte un comportement attendu : il traite bien les objectifs posés et prend en compte leur état courant, il ne prend pas le risque de voir son niveau d'énergie baisser dangereusement, utilise les meilleures outils à sa disposition lorsqu'ils sont disponibles et évite les pénalités autant que possible. Ces résultats démontrent la faculté de l'architecture à traiter des objectifs sous la forme de motivations dans un environnement complexe et de grande taille. Les comportements observés montrent la bonne capacité de prédiction du système, celui-ci n'exécutant pas de tâche pouvant mettre le robot en difficulté dans le futur. Ce test montre enfin la capacité du cycle délibération / exécution à traiter un problème très long.

Ensuite, avec plusieurs configurations, nous avons testé les facteurs limitant le développement de l'arbre des macro-actions afin de vérifier leur efficacité. Nous avons pu constater que les limitations jouent bien leur rôle : en ne développant pas les parties de l'arbre les moins probables, le coût des phases de délibération a bien chuté sans réduire de façon conséquente les performances du système. Cette capacité montre également l'efficacité des prédictions produites, puisque les situations jugées improbables ne ce sont effectivement que très rarement produites.

Nous avons également choisi d'explorer une nouvelle méthode d'exécution de l'agenda

des macro-actions. En stoppant son exécution avant d'atteindre la fin de l'agenda, nous avons pu obtenir des performances légèrement meilleures. Cela nous permet de montrer que les décisions prises en fin d'agenda sont moins précises et moins optimales, car prenant moins en compte leur impact sur les décisions futures.

Les tests expérimentaux effectués nous ont permis de constater que l'architecture proposée est fonctionnelle. Celle-ci fait apparaître des comportements logiques, en satisfaisant au mieux les objectifs posés. De plus, si les performances en termes de récompense accumulée sont difficiles à commenter, nous avons pu voir que le processus de planification est rapide et peu gourmand en ressource informatique pour un problème de taille importante.

En conclusion, nous avons trouvé une solution permettant de résoudre la problématique posée. En proposant un nouveau modèle pour les objectifs, nous avons pu représenter des problèmes complexes. Nous avons pu partir de ces motivations pour créer un processus de planification en deux étapes, en calculant d'abord des politiques permettant de faire progresser les motivations, puis en utilisant ces politiques afin d'effectuer une recherche locale de plus haut niveau. Avec les calculs présentés, nous avons pu obtenir une bonne capacité de prédiction et donc apporter au système la capacité d'être autonome dans le choix de ces objectifs.

7.2 Limitations et ouvertures

Lors du travail que nous avons effectué, il y a plusieurs pistes que nous n'avons pas pu ou eu le temps d'explorer. Pour conclure cette thèse, nous donnons les pistes de recherches que nous aurions pu explorer pour accomplir ce travail ou qui pourraient le poursuivre.

Intégration et tests en situation réelle et nouveaux scénarios. Pendant ce travail de thèse, nous avons pu mener des tests de l'architecture que nous avons proposée en simulation. Pour aller plus loin, il serait intéressant de pouvoir tester le système sur un agent physique, afin de vérifier la faisabilité des solutions proposées, par exemple la définition d'un modèle décrit avec les PSO, et leurs performances en situation réelle. Cela serait également l'occasion de tester de nouveaux scénarios permettant de soulever de nouvelles questions ou de nouvelles exigences sur la définitions des motivations. Enfin et puisque cette thèse se déroulait dans le cadre d'un projet, intégrer notre travail dans une architecture robotique proposant de nouvelles fonctionnalités (apprentissage, découverte du modèle, etc.), dans lequel le système proposé occupera la partie de la décision de plus haut niveau, Son intégration pourra poser de nouveaux défis en termes de définition du modèle et des motivations.

Nouvelles macro-actions. Pour la définition des politiques calculées, nous avons choisi la solution la plus simple, c'est-à-dire ignorer la notion de ressources. Les politiques π^{msv}_{rmt} sont calculées sur la seule base de la définition du modèle du monde sans ressource et leurs consommations en ressources sont estimées dans un second temps. Or, il serait intéressant d'introduire les ressources comme facteurs d'optimisation pour le calcul de ces politiques. En créant plusieurs profils d'optimisation, il serait possible de créer plusieurs versions d'une même macro-action : une version optimisant la dépense d'énergie et une autre exécutant la tâche le plus rapidement possible, par exemple. Cette diversité aura un

coût non négligeable en calcul, multipliant les macro-actions à calculer et complexifiant le développement de l'arbre des macro-actions. En revanche, cette possibilité pourrait permettre à un agent de mieux s'adapter aux aléas lors des expérimentations.

Estimations et contrôle de la consommation des ressources. Durant les recherches effectuées, nous n'avons pas réussi à trouver une méthode exacte permettant d'estimer la courbe de la consommation en ressources des politiques calculées. Cependant, plusieurs pistes de calculs nous ont fait penser qu'une solution plus fiable pouvait être calculée. Dans le cas d'une chaîne de Markov acyclique, il est en effet possible de calculer simplement les bornes inférieures et supérieures. Nous pensons qu'il peut exister une méthode pour créer des points intermédiaires et d'étendre ce calcul aux chaînes comportant des cycles (ce qui est pratiquement toujours le cas dans les problèmes traités). Bénéficier d'une courbe de la consommation des ressources plus fiable permettrait d'améliorer sensiblement la capacité du système à prévoir les scénarios futurs. De plus, il serait intéressant d'explorer des méthodes pour suivre plus efficacement l'évolution du niveau des ressources au cours de l'exécution d'un agenda de macro-action.

Adaptation aux extensions des processus de décision markoviens. Pour la problématique proposée, nous n'avons retenu que l'incertitude dû à l'exécution des actions. Dans cette situation, nous avons développé notre architecture autour des MDPs "simples". Ainsi, il serait intéressant d'étendre ce travail aux cas où l'état du système est incertain (processus de décision markoviens partiellement observables) ou avec la présence de multiples agents.

Évolution des motivations. Lors de la conception de ce travail et lors de expérimentations, nous avons considéré que les motivations définies au départ restaient inchangées tout au long de la résolution du problème. Il serait intéressant d'étudier de tels changements, qui permettraient à un agent de s'adapter ou même de prendre en compte de nouvelles problématiques lors de son fonctionnement. Dans ce cadre, nous pouvons différencier deux types de modifications, légères et lourdes. Les premières concernent les modifications apportées aux valeur de récompense des *rmt* ainsi qu'au changement des seuils en ressources des *rmt*. Pour ces cas, seul le nouveau calcul d'un agenda de macro-actions semble nécessaire pour prendre en compte les changements. Cela pourrait permettre d'ajuster la récompense d'une tâche ou de modifier le poids d'une motivation relativement aux autres. Les modifications lourdes correspondent à toutes les autres modifications : nouvelle motivation, nouvelle *rmt*, changement de la transition but d'une *rmt*. Dans ces cas, une ou plusieurs macro-actions doivent être recalculées afin de s'adapter aux changements.

Construction automatique de motivations. La dernière et peut-être la plus intéressante des pistes concerne la conception automatique des motivations. En effet, nous pouvons constater chez les agents intelligents qu'il peut être assez aisé de constituer les motivations de base. Ces motivations ont pour objectif de maintenir l'agent en état de fonctionner (conservation d'énergie, évitement des causes de dommages, réparation). Cependant, il nous est possible, en connaissant ces motivations et le fonctionnement de l'environnement, de discerner des motivations plus complexes issues de ces motivations de base. Par exemple, la motivation présentée dans le Chapitre 6 consistant pour le robot à

maintenir les bornes de rechargement en état de fonctionner peut être considérée comme issue de la motivation consistant à conserver un niveau d'énergie minimal. Pour imaginer un robot complètement autonome dans un environnement, la capacité à déduire des objectifs à accomplir en fonction de ses besoins semble intéressante, voire essentielle. De plus, la capacité à découper des tâches en sous-tâches pourrait permettre à un agent de mieux découper son investissement en temps, et pourrait lui permettre de diminuer la quantité de calculs à effectuer (grâce à l'utilisation de plusieurs tâches utilisant des sous-modèles spécifiques), et ainsi améliorer ses performances.

Essayer de résoudre cette problématique est donc un enjeu crucial pour développer la capacité d'un agent robotique à être autonome et à survivre dans son environnement.

Références

[Alami et al., 1998] Alami, R., Chatila, R., Fleury, S., Ghallab, M., and Ingrand, F. (1998). An architecture for autonomy. *The International Journal of Robotics Research*, 17(4):315–337.

[Bahar et al., 1997] Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., and Somenzi, F. (1997). Algebric decision diagrams and their applications. *Formal methods in system design*, 10(2-3):171–206.

[Bonet and Geffner, 2001] Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33.

[Boutilier et al., 1999] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11(1):94.

[Boutilier et al., 2000] Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial intelligence*, 121(1):49–107.

[Bradtke and Duff, 1995] Bradtke, S. J. and Duff, M. O. (1995). Reinforcement learning methods for continuous-time markov decision problems. *Advances in neural information processing systems*, pages 393–400.

[Brenner and Nebel, 2009] Brenner, M. and Nebel, B. (2009). Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331.

[Bresina et al., 2002] Bresina, J., Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D., et al. (2002). Planning under continuous time and resource uncertainty: A challenge for ai. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 77–84. Morgan Kaufmann Publishers Inc.

[Carbone et al., 2008] Carbone, A., Finzi, A., Orlandini, A., and Pirri, F. (2008). Model-based control architecture for attentive robots in rescue scenarios. *Autonomous Robots*, 24(1):87–120.

[Ceballos et al., 2011] Ceballos, A., Bensalem, S., Cesta, A., De Silva, L., Fratini, S., Ingrand, F., Ocon, J., Orlandini, A., Py, F., Rajan, K., et al. (2011). A goal-oriented autonomous controller for space exploration. *ASTRA*, 11.

[Chakrabarti, 1999] Chakrabarti, S. K. (1999). Markov equilibria in discounted stochastic games. *Journal of Economic Theory*, 85(2):294–327.

[Chatterjee et al., 2006] Chatterjee, K., Majumdar, R., and Henzinger, T. A. (2006). Markov decision processes with multiple objectives. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 325–336. Springer.

[Clore et al., 2001] Clore, G. L., Gasper, K., Garvin, E., and Forgas, J. P. (2001). Affect as information. *Handbook of affect and social cognition*, pages 121–144.

[Coddington, 2007] Coddington, A. (2007). Motivations as a meta-level component for constraining goal generation. In *In Proceedings of the First International Workshop on Metareasoning in Agent-Based Systems*, pages 16–30.

[Coddington and Luck, 2003] Coddington, A. M. and Luck, M. (2003). Towards motivation-based plan evaluation. In *FLAIRS Conference*, pages 298–302.

[Coddington and Luck, 2004] Coddington, A. M. and Luck, M. (2004). A motivation-based planning and execution framework. *International Journal on Artificial Intelligence Tools*, 13(01):5–25.

[Debreu, 1954] Debreu, G. (1954). Valuation equilibrium and pareto optimum. *Proceedings of the national academy of sciences*, 40(7):588–592.

[Dietterich, 2000a] Dietterich, T. G. (2000a). Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303.

[Dietterich, 2000b] Dietterich, T. G. (2000b). An overview of maxq hierarchical reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 26–44. Springer.

[Doherty et al., 2009] Doherty, P., Kvarnström, J., and Heintz, F. (2009). A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3):332–377.

[Emerson, 1990] Emerson, E. (1990). Temporal and modal logic, chapter handbook of theoretical computer science (vol. b): formal models and semantics. *Cambridge, MA, USA*, 97:9951072.

[Feng et al., 2004] Feng, Z., Dearden, R., Meuleau, N., and Washington, R. (2004). Dynamic programming for structured continuous markov decision problems. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 154–161. AUAI Press.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208.

[Fox and Long, 2003] Fox, M. and Long, D. (2003). Pddl2. 1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)*, 20:61–124.

[Fox and Long, 2006] Fox, M. and Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res.(JAIR)*, 27:235–297.

[Galindo et al., 2008] Galindo, C., Fernández-Madrigal, J.-A., González, J., and Saffiotti, A. (2008). Robot task planning using semantic maps. *Robotics and autonomous systems*, 56(11):955–966.

[Gat et al., 1998] Gat, E. et al. (1998). On three-layer architectures. *Artificial intelligence and mobile robots*, 195:210.

[Georgeff and Lansky, 1987] Georgeff, M. P. and Lansky, A. L. (1987). Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682.

[Ghallab and Laruelle, 1994] Ghallab, M. and Laruelle, H. (1994). Representation and control in ixtet, a temporal planner. In *AIPS*, volume 1994, pages 61–67.

[Guestrin et al., 2006] Guestrin, C., Hauskrecht, M., and Kveton, B. (2006). Solving factored mdps with hybrid state and action variables. *Journal Of Artificial Intelligence Research*.

[Hanheide et al., 2010] Hanheide, M., Hawes, N., Wyatt, J., Göbelbecker, M., Brenner, M., Sjöö, K., Aydemir, A., Jensfelt, P., Zender, H., and Kruijff, G.-J. (2010). A framework for goal generation and management. In *Proceedings of the AAAI workshop on goal-directed autonomy*.

[Haslum and Geffner, 2014] Haslum, P. and Geffner, H. (2014). Heuristic planning with time and resources. In *Sixth European Conference on Planning*.

[Hauskrecht et al., 1998] Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., and Boutilier, C. (1998). Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 220–229. Morgan Kaufmann Publishers Inc.

[Hawes, 2011] Hawes, N. (2011). A survey of motivation frameworks for intelligent systems. *Artificial Intelligence 175*.

[Hung and Leachman, 1996] Hung, Y.-F. and Leachman, R. C. (1996). A production planning methodology for semiconductor manufacturing based on iterative simulation and linear programming calculations. *IEEE Transactions on Semiconductor manufacturing*, 9(2):257–269.

[Ingrand and Ghallab, 2014] Ingrand, F. and Ghallab, M. (2014). Deliberation for autonomous robots: A survey. *Artificial Intelligence*, pages 1–40.

[Ingrand et al., 2007] Ingrand, F., Lacroix, S., Lemai-Chenevier, S., and Py, F. (2007). Decisional autonomy of planetary rovers. *Journal of Field Robotics*, 24(7):559–580.

[Jónsson et al., 2000] Jónsson, A. K., Morris, P. H., Muscettola, N., Rajan, K., and Smith, B. D. (2000). Planning in interplanetary space: Theory and practice. In *AIPS*, pages 177–186.

[Khatib et al., 2001] Khatib, L., Muscettola, N., and Havelund, K. (2001). Mapping temporal planning constraints into timed automata. In *Temporal Representation and Reasoning, 2001. TIME 2001. Proceedings. Eighth International Symposium on*, pages 21–27. IEEE.

[Knight et al., 2001] Knight, S., Rabideau, G., Chien, S., Engelhardt, B., and Sherwood, R. (2001). Casper: Space exploration through continuous planning. *IEEE Intelligent Systems*, 16(5):70–75.

[Kolobov, 2012] Kolobov, A. (2012). Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210.

[Kovacs, 2012] Kovacs, D. L. (2012). A multi-agent extension of pddl3. 1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC), ICAPS-2012, Atibaia, Brazil*, pages 19–27.

[Kress-Gazit et al., 2009] Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. (2009). Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381.

[Kuipers and Byun, 1991] Kuipers, B. and Byun, Y.-T. (1991). A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and autonomous systems*, 8(1):47–63.

[Kushmerick et al., 1995] Kushmerick, N., Hanks, S., and Weld, D. S. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1):239–286.

[Le Gloannec et al., 2008] Le Gloannec, S., Mouaddib, A., and Charpillet, F. (2008). Adaptive multiple resources consumption control for an autonomous rover. In *European Robotics Symposium 2008*, pages 1–11. Springer.

[Lemai and Ingrand, 2004] Lemai, S. and Ingrand, F. (2004). Interleaving temporal planning and execution in robotics domains. In *AAAI*, volume 4, pages 617–622.

[Littman et al., 1995] Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995). On the complexity of solving markov decision problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 394–402. Morgan Kaufmann Publishers Inc.

[Mausam and Weld, 2008] Mausam and Weld, D. S. (2008). Planning with durative actions in stochastic domains. *J. Artif. Intell. Res.(JAIR)*, 31:33–82.

[Mausam et al., 2005] Mausam, M., Benazera, E., Brafman, R., and Hansen, E. (2005). Planning with continuous resources in stochastic domains.

[McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). Pddl-the planning domain definition language.

[McGann et al., 2008] McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., and McEwen, R. (2008). A deliberative architecture for auv control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1049–1054. IEEE.

[Meuleau et al., 2009] Meuleau, N., Benazera, E., Brafman, R. I., Hansen, E. A., and Mausam, M. (2009). A heuristic search approach to planning with continuous resources in stochastic domains. *Journal of Artificial Intelligence Research*, 34(1):27.

[Molineaux et al., 2010] Molineaux, M., Klenk, M., and Aha, D. W. (2010). Goal-driven autonomy in a navy strategy simulation. Technical report, DTIC Document.

[Muscettola et al., 1998] Muscettola, N., Nayak, P. P., Pell, B., and Williams, B. C. (1998). Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1):5–47.

[Nau et al., 2003] Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). Shop2: An htn planning system. *J. Artif. Intell. Res.(JAIR)*, 20:379–404.

[Pettersson, 2005] Pettersson, O. (2005). Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53(2):73–88.

[Pollack and Horty, 1999] Pollack, M. E. and Horty, J. F. (1999). There's more to life than making plans: plan management in dynamic, multiagent environments. *AI Magazine*, 20(4):71.

[Rabideau et al., 1999] Rabideau, G., Knight, R., Chien, S., Fukunaga, A., and Govindjee, A. (1999). Iterative repair planning for spacecraft operations using the aspen system. In *Artificial Intelligence, Robotics and Automation in Space*, volume 440, page 99.

[Renaudo et al., 2015] Renaudo, E., Girard, B., Chatila, R., and Khamassi, M. (2015). Respective advantages and disadvantages of model-based and model-free reinforcement learning in a robotics neuro-inspired cognitive architecture. *Procedia Computer Science*, 71:178–184.

[Saaty, 2008] Saaty, T. L. (2008). Decision making with the analytic hierarchy process. *International journal of services sciences*, 1(1):83–98.

[Sanner, 2011] Sanner, S. (2011). Relational dynamic influence diagram language (rddl): Language description (2010). *URL http://users. cecs. anu. edu. au/˜ ssanner/IPPC_2011/RDDL. pdf. Citado na pág*, 55:59–79.

[Scheutz and Schermerhorn, 2009] Scheutz, M. and Schermerhorn, P. (2009). Affective goal and task selection for social robots. *Handbook of research on synthetic emotions and sociable robotics: new applications in affective computing and artificial intelligence*, page 74.

[Schiffer et al., 2012] Schiffer, S., Ferrein, A., and Lakemeyer, G. (2012). Caesar: an intelligent domestic service robot. *Intelligent Service Robotics*, 5(4):259–273.

[Sloan, 2007] Sloan, T. W. (2007). Safety-cost trade-offs in medical device reuse: a markov decision process model. *Health Care Management Science*, 10(1):81–93.

[Soto et al., 2007] Soto, M., Nava, P., and Alvarado, L. (2007). Drone formation control system real-time path planning. In *AIAA Infotech@ Aerospace 2007 Conference and Exhibit*, page 2770.

[Sutton et al., 1999] Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211.

[Svorenova et al., 2013] Svorenova, M., Cerna, I., and Belta, C. (2013). Optimal control of mdps with temporal logic constraints. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 3938–3943. IEEE.

[Vattam et al., 2013] Vattam, S., Klenk, M., Molineaux, M., and Aha, D. W. (2013). Breadth of approaches to goal reasoning: A research survey. Technical report, DTIC Document.

[Wawerla and Vaughan, 2007] Wawerla, J. and Vaughan, R. (2007). Near-optimal mobile robot recharging with the rate-maximizing forager. *Advances in Artificial Life*, pages 776–785.

[Wilson et al., 2014] Wilson, M. A., McMahon, J., and Aha, D. W. (2014). Bounded expectations for discrepancy detection in goal-driven autonomy. Technical report, DTIC Document.

[Záková et al., 2011] Záková, M., Kremen, P., Zelezny, F., and Lavrac, N. (2011). Automating knowledge discovery workflow composition through ontology-based planning. *IEEE Transactions on Automation Science and Engineering*, 8(2):253–264.