Modèles de décision de groupe pour la coordination et le contrôle d'un système multi-robots

Abdel-Illah Mouaddib*
GREYC-Université de Caen
Maréchal Juin, BP 5186
14032 Caen Cedex
mouaddib@info.unicaen.fr

Résumé

Nous étudions de nouveaux modèles décisionnels pour la conception de systèmes d'agents autonomes évoluant dans un environnement dynamique et incertain. Cette orientation commence à prendre de plus en plus d'ampleur dans la communauté internationale dans les domaines de la planification multi-agent et des systèmes complexes appliqués à plusieurs domaines dont le plus notable est la coordination dans les systèmes multi-robots.

Pour cela nous faisons appel aux processus décisionnels de Markov décentralisés (DEC-MDP). La résolution de DEC-MDP est un problème difficile. Une solution optimale ne peut donc être calculée que pour de très petits problèmes. De nombreux travaux se sont intéressés à la mise en place de techniques d'approximation mais elles ne gèrent que très peu de contraintes sur les problèmes posés. Dans cet article, nous identifions une classe de DEC-MDP, OC-DEC-MDP, pouvant gérer des contraintes temporelles et de précédence entre les tâches. De plus, nous proposons un algorithme polynomial pouvant résoudre cette classe de DEC-MDP. Plusieurs extensions sont actuellement développés pour la prise en compte de la coordination spatiale pour une meilleure planification collective des chemins des robots et la coordination des décisions locales pour la construction d'une décision globale proche de l'optimum.

Mots Clef

Processus Décisionnels de Markov Décentralisés, planification multi-objective, décision, systèmes multi-agents

1 Introduction

Nous étudions de nouveaux modèles décisionnels pour la conception de systèmes d'agents autonomes évoluant dans un environnement dynamique et incertain. Cette orientation commence à prendre de plus en plus d'ampleur dans la communauté internationale dans les domaines de la planification multi-agent et des systèmes complexes appliqués à plusieurs domaines comme les moteurs de recherche adaptatifs, Internet et les réseaux informatiques et la robotique. L'objectif du projet est de formaliser les problèmes de la prise de décision par un groupe d'agents par des processus décisionnels de Markov décentralisés (DEC-MDP). Ces modèles que nous avons utilisé pour le contrÃ´le d'un agent autonome (un robot) dans les domaines de la robotique exploratoire , nous souhaitons les étudier dans un cadre multi-robot en étudiant les problèmes de l'interaction et la coordination entre ces processus, et la communication qui peut être impossible pendant certaines périodes de temps (panne, visibilité, bruit, etc.). Ces modèles seront évalués dans des scénarios de la robotique de combats ou de secours lors d'une situation de crise.

Différents travaux se sont intéressés à l'extension des MDPs et POMDPs pour le contrôle décentralisé de systèmes multi-agents coopératifs. Parmi les approches mises en place, différentes hypothèses concernant l'observabilité des agents et les types de communication ont été envisagées, néanmoins il reste très difficile de résoudre les problèmes considérés. Il a été prouvé que les DEC-MDP et les DEC-POMDP étaient des problèmes NEXP [Bernstein *et al.*2000]. La plupart des travaux tentent donc de trouver de bonnes approximations de la solution optimale. Certaines approches permettent d'atteindre des optima locaux comme le Joint Equilibrium based Search for Policies (JESP) de Nair et Tambe [Yokoo *et al.*2003]. D'autres modèles ont recours à des méthodes heuristiques afin d'approximer la politique optimale : les DEC-POMDP avec communication (DEC-POMDP-Com) proposés par Goldman et Zilberstein [Goldman and Zilberstein2003], le Communicative Multi-agent Team Decision Problem (COM-MTDP) décrit par Pynadath et Tambe [Pynadath and Tambe2002] ; ou bien elles utilisent l'apprentissage on-line comme le Partial Observable Identical Payoff Stochastic Game (POIPSG) par Peshkin et al. [Peshkin *et al.*2000]. L'une des principales difficultés des DEC-MDP et DEC-POMDP réside dans le manque d'observabilité entre les agents : chaque agent ne perçoit que partiellement les autres. Dans les travaux qui viennent d'être cités, les décisions des agents reposent sur les observations locales et les informations collectées en communiquant, à condition que les agents

*Cet article contient du matériel des publications de l'équipe MAD du GREYC

puissent communiquer.

Becker et al. ont identifié deux classes de MDPs pouvant être résolues de manière optimale, par un algorithme exponentiel en espace d'états : Decentralized Decision Process with Event Driven Interaction (ED-DEC-MDP) [Becker *et al.*2004] et les Transition-Independent Decentralized MDPs (TI-DEC-MDP) [Becker *et al.*2003]. Ces approches permettent de gérer des dépendances entre les tâches. Cependant, seuls de très petits problèmes peuvent être traités.

Dans cet article, nous identifions une classe de DEC-MDP, OC-DEC-MDP, pouvant gérer des contraintes complexes et pouvant être résolue par un algorithme polynomial. Les problèmes auxquels nous nous intéressons sont constitués de plusieurs agents ayant un ensemble de tâches à réaliser. Aucune restriction n'est posée concernant le nombre d'agents. Chaque tâche est associée à un agent. Le problème de la répartition des tâches sur les agents n'est pas détaillé dans cet article mais des extensions de cette approche sont proposées pour combler ce besoin.

Chaque tâche possède des contraintes temporelles représentées par une fenêtre temporelle $[EST, LET]$. EST (Earliest Start Time) correspond à la date de début au plus tôt de la tâche. LET (Latest End Time) décrit sa date de fin au plus tard. Chaque tâche t_i possède également des prédécesseurs : les tâches devant être achevées afin que t_i puisse commencer. Par ailleurs, les agents considérés évoluent dans des environnements incertains, différentes durées d'exécution et différentes consommations de ressources doivent donc être envisagées pour chaque tâche. Chaque tâche possède un ensemble de durées d'exécution représentées par une distribution de probabilités, ainsi qu'un ensemble de consommations de ressources également représentées par une distribution de probabilités. Les durées d'exécution et les consommations de ressources sont supposées discrètes. $P_c(\delta_c)_{t_i}$ désigne la probabilité que l'exécution d'une tâche t_i prenne δ_c unités de temps. $Pr(\Delta r)_{t_i}$ désigne la probabilité que l'exécution d'une tâche t_i consomme Δr unités de ressources. Enfin, une récompense est associée à chaque tâche, elle correspond à la récompense obtenue par l'agent lorsqu'il a exécuté la tâche.

L'ensemble des tâches que les agents ont à réaliser est appelé "mission" et peut être représenté par un graphe orienté acyclique. Chaque noeud du graphe correspond alors à une tâche et les arêtes traduisent les contraintes de précédence entre les tâches. Afin de réaliser leur mission, les agents doivent agir de manière coopérative. Durant l'exécution de la mission, les agents ne peuvent pas communiquer et ne peuvent pas observer les états des autres agents. Ils doivent donc coopérer sans communication et tout en gérant l'incertitude sur les actions, les contraintes du problème et le caractère non-observable des autres agents.

La figure 5 présente un exemple de mission pour 3 robots (agents) "rescue" devant gérer une situation de crise. Les robots "rescue" sont des robots devant intervenir lors de tremblements de terre ou d'autres catastrophes du même type. Ils remplacent ou assistent des équipes de secours utilisées traditionnellement. Dans le scénario que nous présentons, des robots pompiers, ambulanciers et policiers doivent agir de manière coopérative afin de gérer une situation de crise. De tels scénari sont inspirés de ceux utilisés lors de la RoboCup Rescue [http ://www.rescuesystem.org/robocuprescue/]. Les arêtes du graphe représentent les contraintes de précédence entre les tâches : les blessés doivent être secourus avant que les pompiers puissent éteindre le feu. Les contraintes temporelles sur les tâches sont mentionnées par des intervalles : l'immeuble a pris feu au temps 3 et les blessés doivent être secourus avant le temps 6. Notre modèle permet de gérer des graphes de tâches beaucoup plus importants, de plus de 200 tâches. D'autres domaines d'applications peuvent également être envisagés : les robots chargés d'explorer d'autres planètes ou des terrains radioactifs, les robots dockers, les satellites, ...

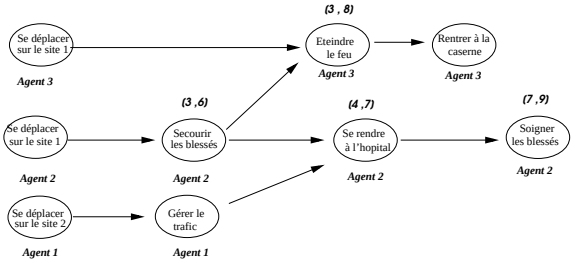

FIG. 1 – Graphe d'une mission

Le problème que nous cherchons à résoudre est un problème de planification multi-agent. Il consiste à déterminer une politique coopérative pour chaque agent. Une politique associe, à chaque état de l'agent, une action. Elle permet alors à celui-ci, lors de l'exécution de la mission, de décider quelle action entreprendre et quand. L'approche que nous présentons se base sur des Processus Décisionnels de Markov décentralisés (DEC-MDP) et utilise la notion de coût occasionné. Les processus décisionnels de Markov décentralisés (DEC-MDP) constituent une extension des processus décisionnels de Markov (MDP) pour les systèmes multi-agents. Ils fournissent un outil mathématique permettant de formaliser et résoudre des problèmes de décision séquentielle dans des systèmes multi-agents où le contrôle est distribué.

La résolution des DEC-MDP étant NEXP, nous proposons de ramener notre problème à un ensemble de MDP locaux. Chaque agent construit son propre MDP local et calcule sa politique locale, en tenant compte des effets de sa décision sur les autres agents. Les contraintes temporelles et de précédence devant être respectées, plus un agent retarde la fin de l'exécution d'une tâche, plus il retarde ses successeurs. Afin de réaliser la mission de manière coopérative, chaque agent doit tenir compte de l'impact de ses décisions sur les autres agents. Pour ce faire, lors du calcul des poli-

tiques des MDPs locaux, l'utilité espérée de chaque action est pondérée par le coût qu'elle occasionne sur les autres agents. Cette approche permet de ne pas avoir recours à des observations sur les états des autres agents, ni à la communication durant l'exécution. Ainsi, la complexité du problème est fortement diminuée.
Nous allons tout d'abord présenter notre approche basée sur l'utilisation de MDP Décentralisés avec Coût Occasionné (OC-DEC-MDP). Nous commencerons par décrire comment construire les différents MDPs locaux constituant le OC-DEC-MDP. Nous présenterons ensuite les différentes équations et l'algorithme permettant de calculer les politiques locales et nous justifierons l'aspect polynomial de notre algorithme. Enfin, nous décrirons différents résultats expérimentaux.

2 Préliminaires

Afin de mettre en place les différents MDPs locaux, il est tout d'abord nécessaire d'avoir de plus amples informations sur la mission. A partir des contraintes temporelles et de précédence ainsi que des durées des tâches, l'ensemble des intervalles d'exécution possibles de chaque tâche peut être déterminé. Des bornes sur les dates de début sont également calculées. Par la suite, LB_i (Lower Bound) désignera la première date de début valide de t_i et UB_i (Upper Bound) la dernière date de début valide. Chacun des intervalles est ensuite pondère par une probabilité. La probabilité associée à un intervalle I d'une tâche t_i représente la probabilité que l'exécution de t_i se déroule dans I. Un algorithme de propagation des probabilités a été mis en place afin de déterminer ces valeurs. Il tient compte des contraintes de précédence et de l'incertitude sur les durées d'exécution.

Probabilités sur les dates de début Afin de calculer les probabilités des intervalles d'une tâche t_i, il est tout d'abord nécessaire de calculer la probabilité de chaque date de début de t_i. La probabilité qu'une tâche commence à st dépend des contraintes de précédence et plus particulièrement des probabilités sur les dates de fin des prédécesseurs. Supposons que la politique initiale d'un agent consiste à commencer l'exécution de ses tâches le plus tôt possible, c'est-à-dire dès que ses prédécesseurs ont terminé. Par conséquent, la probabilité $DP(t)$ que l'agent commence l'exécution d'une tâche t_i à t est égale au produit des probabilités que tous les prédécesseurs aient fini leur exécution avant ou à t et au moins l'un d'entre eux a fini à t (sinon l'exécution de la tâche aurait commencé plus tôt). Plus formellement, pour les racines : $DP(t) = 1, t = LB_{root} = UB_{root}$. Pour les autres noeuds :

$$DP(t) = \prod_{a \in predecessors(t_i)} P^a_{end}(\delta_e \leq t) - \sum_{t_1 < t} DP(t_1)$$

où a est un prédécesseur de t_i dans le graphe de la mission et $P_{end}(\delta_e \leq t)$ est la probabilité que le prédécesseur a finisse à t ou avant. $DP(t)$ est appelé "probabilité absolue". Supposons maintenant que l'agent décide de commencer l'exécution de t_i à t même s'il aurait pu commencer avant (t n'est pas obligatoirement la date de début au plus tôt). La probabilité que l'agent puisse commencer à t est égale à la probabilité que les prédécesseurs aient fini à t ou avant, soit : $\sum_{t' \leq t} DP(t')$. Cette probabilité est appelée "probabilité relative" car elle est relative à une décision de commencer l'exécution à une certaine date t pouvant être différente de la première date de début possible, et par conséquent différente de la date de début dictée par la politique initiale.
Etant donné une tâche t_i, $P^{abs}_w(I)$ désignera la probabilité que t_i s'exécute dans l'intervalle I sachant que l'agent suit sa politique initiale. $P^{rel}_w(I)$ désignera la probabilité que t_i s'exécute dans l'intervalle I sachant que l'agent décide de commencer à st, cette probabilité est relative à la décision prise par l'agent qui peut être différente de celle dictée par sa politique initiale.
La probabilité $P^a_{end}(\delta_e \leq t)$ que le prédécesseur a ait fini à t est définie par :

$$P^a_{end}(\delta_e \leq t) = \sum_{I_1 \in intervalles(a), et(I_1) \leq t} P^{abs}_w(I_1)$$

Elle est égale à la somme des probabilités que le prédécesseur a soit exécuté dans un intervalle I_1 telle que la date de fin $et(I_1)$ de I_1 soit inférieure à t.
Connaissant les probabilités sur les dates de début des tâches, il est possible de calculer les probabilités sur les intervalles d'exécution.

Probabilités des intervalles d'exécution La probabilité qu'une tâche t_{i+1} s'exécute dans un intervalle $I = [st, et]$ est égale à la probabilité que t_{i+1} commence à st et que son exécution dure $\Delta t = et - st$ unités de temps. Par ailleurs, un agent ne peut commencer t_{i+1} que s'il a fini sa tâche précédente t_i. Par conséquent, il connaît la date de fin $et(I')$ de t_i, lorsqu'il commence l'exécution de t_{i+1}. De plus, la date de fin de t_i influence la date de début de t_{i+1} et la probabilité des intervalles d'exécution de t_{i+1}. C'est pourquoi, lors du calcul de la probabilité de chaque intervalle, nous devons tenir compte de la date de fin $et(I')$ de t_i. Pour ce faire, nous calculons les probabilités $P^{abs}_w(I|et(I')_{t_i})$ et $P^{rel}_w(I|et(I')_{t_i})$ que t_{i+1} s'exécute dans l'intervalle I sachant que t_i s'est terminée à $et(I')$. $P^{abs}_w(I|et(I')_{t_i})$ suppose que l'agent suit sa politique initiale, et $P^{rel}_w(I|et(I')_{t_i})$ suppose que l'agent décide de commencer à $st(I)$.

$$P^{abs}_w(I|et(I')_{t_i}) = DP(st(I)|et(I')_{t_i}).P_c(et(I) - st(I))$$

$$P^{rel}_w(I|et(I')_{t_i}) = \sum_{t' \leq st(I)} DP(t'|et(I')_{t_i}).P_c(et(I) - st(I))$$

La probabilité $DP(st(I)|et_{t_i}(I'))$ utilisée ci-dessus correspond à la probabilité que les prédécesseurs de la tâche aient fini à $st(I)$ et au moins l'un d'eux a fini exactement à $st(I)$, sachant que t_i s'est terminée à $et(I')$:

$$DP(st(I)|et(I')_{t_i}) = \prod_{a \in predecessors(t_{i+1}) - t_i} P^a_{end}(\delta_e \leq st(I)|et(I')_{t_i}) - \sum_{t_1 < st(I')} DP(t_1|et(I')_{t_i})$$

$$P^a_{end}(\delta_e \leq t | et(I')_{t_i}) = \sum_{I_1 \in intervalles(a), et(I_1) \leq t} P^{abs}_w(I_1 | et(I')_{t_i})$$

3 Formalisation du problème

Une fois les intervalles d'exécution et leurs probabilités déterminées, il est possible de construire et de résoudre le OC-DEC-MDP correspondant à notre problème.

3.1 Modèle OC-DEC-MDP

A partir du graphe de la mission, il est possible de construire le graphe des tâches de chaque agent A_i. Ce dernier ordonne les tâches que l'agent doit accomplir. Il est également utilisé lors de la construction du OC-DEC-MDP. La figure 2 présente les graphes des tâches des agents et le OC-DEC-MDP, associés à la mission décrite dans la figure 5.

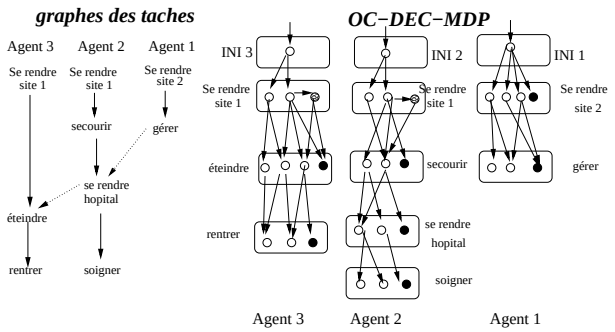

FIG. 2 – Relations entre les graphes des tâches et le OC-DEC-MDP

Un OC-DEC-MDP pour n agents est un ensemble de n MDPs locaux. Chaque MDP local d'un agent A_i est défini par le tuple $< S_i, T_i, P_i, R_i >$ où :

- S_i est l'ensemble fini des états de l' agent A_i
- T_i est l'ensemble fini des tâches que l'agent A_i doit exécuter
- P_i est la fonction de transition telle que $P(s'_i | s_i, t_i)$ est la probabilité d'arriver dans l'état s'_i lorsque A_i exécute la tâche t_i à partir de l'état s_i
- R_i est la fonction de récompense telle que $R_i(t_i)$ est la récompense obtenue par A_i lorsqu'il a exécuté t_i.

Nous allons maintenant détailler chacune des composantes des MDPs locaux.

Etats A chaque étape de décision deux situation peuvent se présenter. Soit, l'agent A_i vient d'exécuter une tâche t_i pendant un intervalle I et il lui reste r unités de ressources. L'état de l'agent est alors un triplet $[t_i, I, r]$ où t_i est la dernière tâche exécutée, I l'intervalle d'exécution de t_i et r la quantité de ressources disponible. L'agent peut aussi avoir essayé d'exécuter une tâche t_{i+1} mais avoir échoué car les prédécesseurs de t_{i+1} n'avaient pas terminé. L'agent est alors dans un état $[t_i, et(I'), I, r]$ tels que t_i est la dernière tâche exécutée par A_i avant qu'il tente d'exécuter t_{i+1} et $et(I')$ est la date de fin de $t_i.I$ désigne l'intervalle pendant lequel l'agent à tenté d'exécuter t_{i+1} et r est la quantité de ressources disponible. A chaque étape, l'agent a une connaissance complète de son état, ce dernier est donc localement totalement observable. En revanche, l'agent ne connaît pas les états des autres agents. Chaque agent observe de manière indépendante son état : les observations d'un agent ne dépendent pas des observations des autres agents.

Tâches - Actions A chaque étape de décision, l'agent doit choisir quelle tâche exécuter et quand. Les actions sont donc de la forme $E(t_i, st)$: exécuter la tâche t_i à st. Les durées et les consommations de ressources étant incertaines, les actions sont non déterministes.

Transitions A partir d'un état $[t_i, I'', r]$ ou $[t_i, et(I''), I, r]$, si un agent décide d'exécuter une tâche t_{i+1} à st, cinq transitions différentes sont envisageables :

- *Exécution réussie :* L'exécution se déroule sans problème, l'agent arrive dans un état sûr $[t_{i+1}, I, r']$. La tâche t_{i+1} a été exécutée dans l'intervalle I' tout en respectant les contraintes temporelles EST et LET de la tâche. $r' = r - \Delta r$ est la quantité de ressources restant après l'exécution de la tâche. La probabilité P_1 d'arriver dans un tel état est définie de la façon suivante :

$$P_1 = \sum_{r \geq \Delta r} \sum_{et(I') \leq LET} P_r(\Delta r).P_w(I' | et(I)_{t_i})$$

- *Date de début invalide ($st \leq UB$) :* L'exécution de la tâche a commencé trop tôt, c'est à dire avant que les prédécesseurs de la tâche aient terminé leur exécution. L'agent arrive donc dans un état d'échec partiel $[t_i, et(I''), [st, st+1], r']$ (la dernière tâche exécutée t_i s'est terminée à $et(I'')$). Cet échec est partiel car l'agent va pouvoir ré-essayer d'exécuter la tâche t_{i+1} par la suite. Nous supposons que l'agent réalise immédiatement (à $st+1$) que les prédécesseurs de la tâche n'ont pas terminé. La probabilité d'arriver dans un tel état est égale à la probabilité $P_{not_end}(st)$ que les prédécesseurs n'aient pas terminé à st et que l'agent ait assez de ressources pour s'en apercevoir. $P_{not_end}(st)$ est égale à la probabilité qu'ils finissent après st ou ne finissent jamais, soit :

$$P_{not_end}(st) = (\prod_{a \in predecessors(t_{i+1}) - t_i} \sum_{I_a \ : et(I_a) > LET - min\delta_i} P_w(I_a | et(I)_{t_i}) - \sum_{t' \leq st} DP(t' | et(I)_{t_i})) +$$
$$(1 - \prod_{a \in predecessors(t_{i+1}) - t_i} \sum_{I_a} P_w(I_a | et(I)_{t_i}))$$

où $DP(t' | et(I)_{t_i})$ est la probabilité que t_{i+1} commence à t' sachant que t_i s'est achevée à $et(I)_{t_i}$.

La probabilité d'arriver dans l'état d'échec partiel $[t_i, et(I''), [st, st+1], r']$ est alors définie de la façon suivante : $P_2 = P_{not_end}(st).P(\Delta r' \leq r)$ sachant que $\Delta r'$, la quantité de ressources consommée pour s'apercevoir que les prédécesseurs n'ont pas terminé, est telle que $r \geq \Delta r'$. $P(\Delta r' \leq r) = 1$ si $\Delta r' \leq r$, sinon $P(\Delta r' \leq r) = 0$

– *Manque de ressources :* Si l'exécution de la tâche consomme plus de ressources que la quantité disponible ou que les prédécesseurs de la tâche n'ont pas terminé et l'agent n'a pas assez de ressources pour s'en apercevoir ($\Delta r' > r$), alors l'agent arrive dans un état d'échec total $[failure_{t_{i+1}}, [st, +\infty], 0]$. La probabilité d'une telle transition est :

$$P_3 = \Big(\sum_{t' \leq st} DP(t'|et(I)_{t_i}) \sum_{r<\Delta r} P_r(\Delta r)$$

$$+P_{not_end}(st).P(\Delta r' > r)\Big)$$

$P(\Delta r' > r) = 1$ si $\Delta r' > r$, sinon $P(\Delta r' > r) = 0$

– *Date de début invalide ($st > UB$) :* Si la tâche commence trop tard, c'est-à-dire après sa date de début au plus tard, l'agent arrive dans un état d'échec total $[failure_{t_{i+1}}, [st, +\infty], r]$. Ce type de transition ne doit être envisagé que si la date de début st de la tâche est telle que $st > UB$, sinon la probabilité d'une telle transition est nulle, $P_4 = 0$. Si $st > UB$, La probabilité d'une telle transition est : $P_4 = P_{not_end}(st)$

– *Violation deadline :* Si l'agent commence l'exécution de la tâche à st mais que la durée d'exécution $\delta_c^{t_{i+1}}$ de la tâche est telle que $st + \delta_c^{t_{i+1}} > LET$, alors l'agent arrive, en raison de la violation d'une des contraintes temporelles, dans un état d'échec total $[failure_{t_{i+1}}, [st, +\infty], r]$. La probabilité associée à une telle transition est :

$$P_5 = \sum_{r \geq \Delta r} \sum_{st+\delta_c^{t_{i+1}} > LET} P_r(\Delta r).DP(st|et(I)_{t_i}).P_c(\delta_c^{t_{i+1}})$$

Il peut facilement être démontré que $P_1 + P_2 + P_3 + P_4 + P_5 = 1$. Ainsi, le système de transition est complet.

Récompenses En fonction de la tâche exécutée, chaque agent A_i reçoit une récompense R_i. La fonction de récompense $R_i(t_i)$ est supposée connue.
Une fois les MDPs locaux mis en place, il est possible de calculer la politique locale de chaque agent.

4 Résolution du problème

Afin de déterminer les politiques des agents nous avons mis en place un algorithme basé sur les principes de la programmation dynamique, et utilisant une équation dérivée de l'équation de Bellman couramment utilisée dans les processus décisionnels de Markov. Nous faisons également appel au coût occasionné, afin que les agents suivent des politiques coopératives. Dans cette section, nous détaillerons les différents aspects du calcul des politiques des agents.

4.1 Evaluation des politiques

Equations de Bellman La politique locale de chaque agent est calculée à l'aide de deux équations. La première permet de déterminer la valeur V d'un état s. Elle correspond à une équation de Bellman usuelle :

$$V[t_i, I, r] = \overbrace{R_i(t_i)}^{\text{gain immédiat}} + \overbrace{max_{E(t_{i+1},s),st>t}(V')}^{\text{Utilité espérée}} \quad (1)$$

La seconde équation est utilisée afin de déterminer la politique Π des agents. Elle correspond à une équation de Bellman dans laquelle nous avons introduit un coût occasionné (OC) :

$$\Pi([t_i, I, r]) = argmax_{E(t_{i+1},st),st>t}\Big(\overbrace{V'}^{\text{Utilité espérée}} - \overbrace{OC(t_{i+1}, st)}^{\text{OC espéré}} \Big) \quad (2)$$

où $OC(t_{i+1}, st)$ est le coût occasionné espéré que l'exécution de t_{i+1} peut provoquer si elle commence à st.
Quelle que soit l'équation utilisée (1 ou 2), V' est tel que : $V' = V1 + V2 + V3 + V4 + V5$. Il existe cinq types de transitions, chacun des termes de V' correspond à l'un d'entre eux. On a alors :

– *Exécution réussie :* $V1 = P_1.V([t_{i+1}, I', r - \Delta r])$

– *Date de début invalide ($st \leq UB$) :*

$$V2 = P_2.V([t_i, et(I'), [st, st+1], r - \Delta r'])$$

– *Ressources insuffisantes :*

$$V3 = P_3.V([failure_{t_{i+1}}, [st, +\infty], 0])$$

– *Date de début invalide ($st > UB$) :*

$$V4 = P_4.V([failure_{t_{i+1}}, [st, +\infty], r])$$

– *Violation deadline :*

$$V5 = P_5.V([failure_{t_{i+1}}, [st, +\infty], r])$$

La valeur d'un état d'échec total $V^*_{fail(t_i)}$ associé à la tâche t_i est donnée par l'équation suivante :

$$V^*_{fail(t_i)} = -R_{t_i} - \sum_{suiv \in agent(t_i)} R_{suiv}$$

où $-R_{t_i}$ est la pénalité immédiate pour avoir échoué totalement, $-\sum_{suiv \in agent(t_i)} R_{suiv}$ représente la perte future de l'agent, elle correspond à la somme des récompenses que l'agent aurait pu obtenir s'il avait eu la possibilité d'exécuter toutes les tâches qu'il lui restait à réaliser.
Comme le montre l'équation 2, la meilleure action à exécuter dans un état s est déterminée en considérant :

– l'utilité espérée de l'agent, calculée à l'aide d'une équation de Bellman (eq. 1). Elle prend en compte ce que l'agent peut espérer gagner en exécutant les tâches qu'il lui reste à réaliser.

– le coût occasionné espéré provoqué sur les autres agents. L'exécution de la prochaine tâche t_{i+1} va en effet provoquer un certain OC sur les autres agents dont certaines tâches sont des successeurs directs ou indirects de t_{i+1}. L'exécution de t_{i+1} va donc influencer l'exécution de leurs tâches.

Comme nous l'expliquerons par la suite, le OC doit être pris en compte si on souhaite déterminer le gain réel d'une action. Ce dernier est égal à l'utilité de la tâche moins son coût occasionné espéré. La meilleure action à exécuter dans un état s est celle qui maximise le gain réel de l'agent.

Coût Occasionné Le coût occasionné est un terme provenant des sciences économiques. La théorie du coût occasionné montre que toute décision a un coût caché, et ignorer ce coût peut conduire à un raisonnement erroné.
Dans notre modèle, le OC est utilisé afin que chaque agent prenne en compte le coût de ses actions sur les autres agents. Tout agent dépense du temps pour exécuter une tâche t_{i+1} et entraîne un OC sur les autres agents. Ce OC correspond à la perte en utilité des autres agents, due au fait que l'exécution de certaines de leurs tâches est retardée par l'exécution de t_{i+1}.
Supposons qu'on ait deux tâches t_{i+1} et t_j exécutées par deux agents différents et telles que t_j soit un successeur de t_{i+1}. La figure 3 représente deux exécutions possibles de t_{i+1}. La première commence à $st1$ et termine à $et1$, la seconde commence à $st2$ et termine à $et2$. Les bornes LB_j et UB_j sur les dates de début de t_j sont également représentées. Lorsque t_{i+1} termine à $et1$, t_j peut commencer son exécution dans l'intervalle $[LB_j, UB_j]$. Soit st_j la meilleure date de début pour t_j dans cet intervalle. En revanche, si t_{i+1} termine à $et2$, l'intervalle des dates de début possibles pour t_j est restreint à $[LB_j + \Delta t, UB_j]$ où $\Delta t = et2 - LB_j$. Soit st'_j la meilleure date de début dans l'intervalle $[LB_j + \Delta t, UB_j]$. Si $st_j = st'_j$, alors le fait que t_{i+1} termine plus tard ne change pas la meilleure date de début pour t_j et le OC est nul. Sinon ($st_j \neq st'_j$), l'utilité espérée V^{st_j} lorsque t_j commence à st est supérieure à l'utilité espérée $V^{st'_j}$ lorsque t_j commence à st'_j (sinon on aurait $st_j = st'_j$). Le fait que t_{i+1} finisse à $et2$ provoque alors un coût occasionné sur l'agent exécutant t_j.

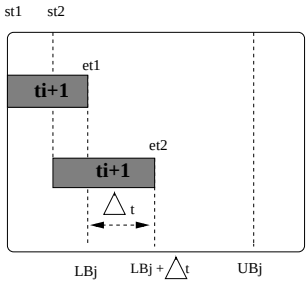

FIG. 3 – Coût occasionné

Le OC sur un agent se traduit donc par la perte en utilité provoquée par un retard Δt. Elle est égale à la différence entre $V^{*0}_{t_j}$, l'utilité espérée de l'agent lorsqu'il commence l'exécution de t_j dans l'intervalle $[LB_j, UB_j]$, et $V^{*\Delta t}_{t_j}$ l'utilité espérée de l'agent lorsqu'il commence l'exécution de t_j dans l'intervalle $[LB_j + \Delta t, UB_j]$. On a alors :

$$OC_{t_j}(\Delta t) = V^{*0}_{t_j} - V^{*\Delta t}_{t_j} \quad (3)$$

où : $V^{*\Delta t}_{t_j} = \sum_{\Delta t^{t_j}_c} P_c(\delta^{t_j}_c).V[t_j, [st_j, st_j + \delta^{t_j}_c], r^{max}_{t_j}]$ st_j est la meilleure date de début pour t_j appartenant à l'intervalle $[LB_j + \Delta t, UB_j]$. $[t_j, [st_j, st_j + \delta^{t_j}_c], r^{max}_{t_j}]$ décrit les états pouvant être atteints lorsque l'exécution de t_j commence à $st_j \in [LB_j + \Delta t, UB_j]$ et dure $\delta^{t_j}_c$ unités de temps. Si $st_j + \delta^{t_j}_c > LET$, l'exécution échoue totalement, on a alors :

$$V[t_j, [st_j, st_j + \delta^{t_j}_c], r^{max}_{t_j}] = V(failure(t_j))$$

Si $\Delta t > UB_{t_j} - LB_{t_j}$ l'exécution ne peut commencer avant la dernière date de début UB_{t_j}, les contraintes temporelles sont violées et on a : $V^{*\Delta t}_{t_j} = -\Big(R(t_j) + \sum_{a \in AllSucc(t_j)} R(a)\Big)$ Sinon, $V[t_j, [st_j, st_j t + \delta^{t_j}_c], r^{max}_{t_j}]$ est calculée à partir de l'équation de Bellman (eq. 1).
Pour chaque tâche t_j, une valeur de OC est calculée pour chaque retard Δt. Pour une tâche, le nombre de valeurs de OC à calculer, dans le pire cas, est égal à $UB_{t_j} - LB_{t_j}$.
Lors du calcul du OC, les ressources des successeurs sont considérées comme maximum. S'ils disposent de moins de ressources (que le maximum possible), le OC provoqué par A_i sera sur-estimé.

$$r^{max}_{t_j} = r_{ini} - \sum_{t_k \in Pred(t_j)} min_{t_k}(\Delta_r)$$

où $Pred(t_j)$ est l'ensemble des tâches exécutées par l'agent avant t_j, et $min_{t_k}(\Delta_r)$ est la consommation minimale de ressources de t_k. Afin de remédier à cette surestimation, il est possible d'avoir recours à un OC multicritères c'est-à-dire dépendant du temps et des ressources ($OC(\Delta t, r)$).
Le OC qui vient d'être décrit, suppose que nous sachions exactement quel est le retard provoqué par t_{i+1} sur les autres agents. Cependant, les actions des agents ne sont pas déterministes donc lorsqu'un agent décide de commencer une tâche à un instant st, il ne sait pas avec certitude quelle va être sa date de fin. C'est pourquoi le OC provoqué par un agent A_i sur les autres agents, lorsqu'il commence l'exécution d'une tâche t_{i+1} à st est un coût occasionné espéré et doit tenir compte des différentes transitions possibles et de leurs probabilités. Le OC espéré provoqué par un agent lorsqu'il décide de commencer une tâche t_{i+1} à st est tel que :

$$\begin{aligned} OC(t_{i+1}, st) = P1 \times & \sum_{ag \in \text{autres_agents}} OC_{ag,t_{i+1}}(et_{i+1}) \\ +(P3 + P4 + P5) \times & \sum_{ag \in \text{autres_agents}} OC_{ag,t_{i+1}}(fail) \\ & +P2 \times OC(t_{i+1}, s = next_start) \end{aligned} \quad (4)$$

où $autres_agents$ est l'ensemble des agents privé de l'agent exécutant t_{i+1}, et_{i+1} est une date de fin de t_{i+1}, $OC_{ag,t_{i+1}}(et_{i+1})$ est le OC espéré provoqué sur l'agent ag lorsque A_i termine l'exécution de t_{i+1} à et. $OC(t_{i+1}, s = next_start)$ est le OC espéré si l'exécution de t_{i+1} échoue partiellement et que A_i ré-essaye d'exécuter la tâche à s (la prochaine date de début d'après la politique calculée). Chaque type de transition doit être pris en compte dans le coût occasionné espéré $OC(t_{i+1}, st)$.
Par ailleurs, si l'agent A_i termine t_{i+1} à $et_{t_{i+1}}$, il n'est pas certain que la prochaine tâche t_j exécutée par ag va commencer à $et_{t_{i+1}}$. Si t_j est un successeur direct de t_{i+1}, t_j

peut être retardée par d'autres prédécesseurs qui vont finir encore plus tard. Dans ce cas, ce n'est pas t_{i+1} qui retarde l'exécution de t_j mais les autres prédécesseurs. Par ailleurs, si t_j n'est pas un successeur direct de t_{i+1}, il existe d'autres tâches t_k entre t_{i+1} et t_j dans le graphe. Le retard provoqué par t_{i+1} peut alors être compensé ou accentué par les tâches t_k. C'est pourquoi le coût occasionné espéré provoqué par t_{i+1} tient compte, pour chaque date de début st_{t_j} de t_j, de la probabilité que t_j commence à st_{t_j} quand t_{i+1} s'est terminée à $et_{t_{i+1}}$. Il est alors calculé par l'équation suivante :

$$OC_{ag,t_{i+1}}(et_{t_{i+1}}) = \sum_{st \in ST(t_j)} DP(st_{t_j}, et_{t_{i+1}}, t_{i+1}) \times OC_{t_j}(st_{t_j} - LB) + (1 - \sum_{st \in ST(t_j)} DP(st_{t_j}, et_{t_{i+1}}, t_{i+1})) \times OC_{t_j}(fail) \quad (5)$$

où t_j est la prochaine action la plus proche dans le graphe qui va être exécutée par ag. $DP(st_{t_j}, et_{t_{i+1}}, t_{i+1})$ est la probabilité que t_j commence à st_{t_j} quand t_{i+1} s'est terminée à $et_{t_{i+1}}$. De plus, $OC_{ag,t_{i+1}}(fail) = OC_{t_j}(fail)$.

4.2 Calcul d'une politique

Algorithme de résolution. Afin d'évaluer chacun des états des agents et de calculer leurs politiques locales, un algorithme basé sur les principes de la programmation dynamique, a été mis en place. En raison des dépendances entre les agents, tous les MDPs locaux doivent être évalués en parallèle. L'ensemble des états du OC-DEC-MDP est constitué de l'union des états des MDPs locaux. Il est organisé en niveaux. Le premier niveau L_0 contient la racine du graphe de la mission. Le niveau L_n contient les successeurs des tâches du niveau L_{n-1}. L'algorithme parcourt chaque niveau à partir du dernier et jusqu'au premier, et évalue les états associés aux tâches du niveau courant. Le pseudo-code de l'algorithme est le suivant :

1. **Pour chaque niveau** L_n des feuilles à la racine du graphe
2. **Pour chaque** tâche t_i appartenant au niveau L_n
3. • Calculer la valeur V de l'état d'échec total : $[l_{i-1}, [*, +\infty], *]$
4. **Pour chaque** date de début st de UB_{t_i} à LB_{t_i}
5. **Pour chaque** taux de ressources r_{t_i} associé à un état d'échec partiel
6. • Calculer la valeur V et la politique Π associées à l'état d'échec partiel : $[t_i, [st, st+1], r_{t_i}]$
7. **fin pour**
8. **Pour chaque** durée Δt_1 de t_i
9. **Pour chaque** taux de ressources r_{t_i} associée à une exécution correcte de t_i
10. • Calculer la valeur$V$ et la politique Π associées aux états : $[t_i, [st, st+\Delta t_1], r_{t_i}]$
11. **fin pour**
12. **fin pour**
13. • Calculer $V_k^{*\Delta t}$ où $\Delta t = st - LB_{t_i}$
14. **fin pour**
15. **Pour chaque** $V_k^{*\Delta t}$ calculé précédemment
16. • Calculer $OC(\Delta t) = V_k^{*0} - V_k^{*\Delta t}$
17. **fin pour**
18. **fin pour**
19. **fin pour**

L'évaluation de chaque état $s = [t_i, r, I]$ est réalisée à l'aide de l'équation de Bellman présentée précédemment. La politique en s est déterminée à l'aide de l'équation 2.
Cet algorithme peut être exécuté de manière centralisée ou décentralisée. L'évaluation des états et le calcul des politiques sont réalisés off-line, c'est-à-dire avant l'exécution de la mission. Dans le cas d'une exécution centralisée tous les états du OC-DEC-MDP sont évalués par une entité centrale, puis chaque agent reçoit sa politique locale. Dans le cas d'une exécution décentralisée, chaque agent évalue ses propres états et communique les valeurs de coût occasionné de ses tâches, aux autres agents. Chaque agent connaît alors son graphe de tâches, il n'a pas besoin de connaître l'ensemble du graphe de la mission. Dans tous les cas, le calcul de la politique locale d'un agent ne requiert pas la connaissance des politiques des autres agents, seul le OC sur les autres agents doit être connu. L'exécution de la mission (on-line) est toujours décentralisée. Chaque agent ne connaît que sa propre politique locale et ne peut observer les états des autres agents. Durant l'exécution, les agents ne communiquent pas de façon directe. Chaque agent peut obtenir des informations de manière indirecte sur l'état de ses prédécesseurs. Un agent sait si les prédécesseurs d'une tâche t_i ont terminé, quand il commence à exécuter t_i.

Etude de la complexité. Afin de prouver l'efficacité de notre approche, nous nous sommes intéressés à la complexité, dans le pire cas, des OC-DEC-MDP. Cette complexité repose sur la taille de l'espace d'états du problème, et par conséquent dépend des paramètres suivants : le nombre maximum de tâches par agent $\#n_{tasks}$, le nombre maximum d'intervalles d'exécution par tâche $\#n_{Max_Interv}$, le nombre maximum de dates de début pour une tâche $\#_{Max_Date_Deb}$, et le nombre maximum de taux de ressources pour chaque tâche $\#n_{Max_Res}$. La taille de l'espace d'état dans le pire cas, est exprimée par l'équation suivante : $\#n_{tasks} \times (\#n_{Max_Interv} + \#_{Max_Date_Deb}) \times \#_{Max_Res}$ L'algorithme de résolution d'un OC-DEC-MDP est polynomial en espace d'états. **Preuve :** L'algorithme décrit précédemment permet de résoudre tout OC-DEC-MDP. Il parcourt et évalue tous les états des MDPs locaux. Le calcul de la valeur V d'un état et la détermination de la politique Π^* ont une complexité en $O(1)$. Les lignes 4 à 14 évaluent et calculent la politique de tous les états associés à la tâche t_i. La complexité de cette phase est en $O((\#n_{Max_Interv} + \#_{Max_Date_Deb}).\#n_{Max_Res})$. Les lignes 15 à 17 calculent les valeurs du OC pour chaque retard possible pour t_i. Dans le pire cas, le nombre de retards possibles est égal au nombre de dates de début possibles pour t_i soit $UB - LB$. La complexité

de cette partie de l'algorithme est en $O(UB - LB)$. Les lignes 4 à 17 sont exécutées pour chaque tâche. Par ailleurs, $O(UB - LB) << O((\#n_{Max_Interv} + \#_{Max_Date_Deb}).\#n_{Max_Res})$. La complexité globale de l'algorithme est donc en $O(\#n_{tasks}.(\#n_{Max_Interv} + \#_{Max_Date_Deb}).\#n_{Max_Res})$ c'est-à-dire en $O(|S|)$. Par conséquent, l'algorithme est polynomial en espace d'états.

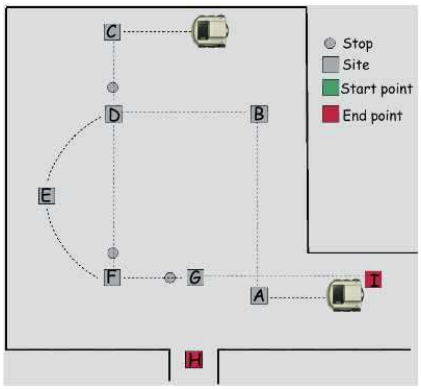

FIG. 4 – Une mission réalisée par 2 rescue robots

The problem can be represented using a graph of tasks. Each node of the graph is a task and edges stand for precedence constraints. Figure 5 illustrates this graphical representation with a space mission scenario involving two robots that have to explore a set of sites. This scenario is described on Figure 4. Exploration consists of different kinds of measurements. Precedence constraints has to be considered : the first robot must snap the sites before the other robot could perform measurements. This mission involves temporal constraints, for instance measurements of site A cannot start before 5 and must be finished at 9.

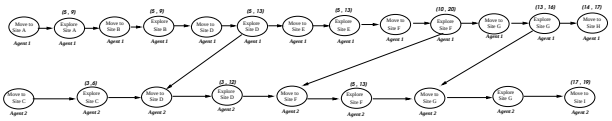

FIG. 5 – Un graphe de mission de 2 rescue robots

5 Exploration Scenario

Le scénario décrit dans la Figure 5 a été implémenté en utilisant des robots Koala. Les robots doivent se coordonner les visites aux sites D, F et G : le premier robot doit quitter le site afin de laisser le second y entrer. Comme montré dans la figure Figure 6, la coordination a fonctionné sans que ls robots ont eu à communiquer. Les contraintes temporelles comme temporelles ont ét é respectées. Les robots prennent des décisions sur le moment de commencer une tâche en tenant compte le fait que d'autres robots attendent (coût occasionné). La décision du second robot est fondée sur la probabilité que les autres robots ont quitté le site, le coût de l'échec partiel et son gain espéré.

6 Extensions : Coordination Spatiale et Allocation dynamique de tâches

Dans le travail ci-dessus décrit, nous nous donnons un graphe de tâches et une allocation des tâches aux robots. Dans cette section, nous présentons quelques techniques permettant l'allocation des tâches et la coordination spatiale permettant une meilleure répartition des tâches sur les agents. En effet, les robots évoluent dans un environnement dans lequel des tâches sont à effectuer. Ces tâches peuevent être inter-dépendantes. Cette dépendance qu'on a jusqu'à maintenant représenter par une contrainte de précédence mais qu'on souhaite étendre à d'autres types de dépendances comme les dépendances entre qualités de réalisation des tâches : la rálisation d'une tâche dégrade, améliore ou interdit la réalisation d'une autre tâche sont des dépendances que nous souhaitons intégrer dans notre approche. Ces types des dépendances sont importantes dans la coordination spatiale car le déplacement d'un robot vers une zone peut rendre le déplacement des autres robots facile, difficile ou impossible. La décision locale d'un robot de se diriger vers cette zone doit tenir compte de ces trois critères : le bénéfice individuel, le bénéfice collectif et les dégâts provoqués par une décision. Cette formalisation mène vers un problème de décision multi-critères [Mouaddib2004] que nous présentons ci-dessous.

6.1 MDP multicritères pour la coordination spatiale

Le formalisme adopté est un DEC-MDP multi-critère donné par le n-uplet $< S_i, A_i, AR_i, P_i, \{FB_{ij}, LB_{ij}, F_{ij}, L_{ij}\} >$:

- S_i est l'espace d'états du robot i, qui représente les tâches effectuées par le robot i comme précédemment.
- A_i est l'ensemble des actions.
- AR_i est une fonction de récompense augmentée par la satifaction individuelle, la satifaction du groupe et les penalités provoqués sur le groupe. Cette fonction est représentée par un vecteur à trois diemnsions que nous détaillerons ci-dessous.
- P_i une distribution de probabilités comme ci-dessus.
- Enfin, l'ensemble $\{FB_{ij}, LB_{ij}, F_{ij}, L_{ij}\}$ qui exprime toutes les dépendances qu'il peut y a voir entre les tâches. Ainsi pour une tâche t_{ij} d'un robot i, on associe 4 ensembles FB_{ij} qui contient les tâches qui améliorent la réalisation de la tâche t_{ij}, LB_{ij} qui contient les tâches qui dégradent la réalisation de la tâche t_{ij}, F_{ij} qui contient les tâches dont la réalisation est améliorée par la réalisation de la tâche t_{ij}, et L_{ij} qui contient les tâches dont la réalisation est dégradée par la réalisation de la tâche t_{ij}. L'état de ces 4 ensembles a une incidence sur la fonction AR_i que nous expliquons ci-dessous. La fonction de récompense augmenté $AR_i(s - i^t)$ est un vecteur donné par le triplet :

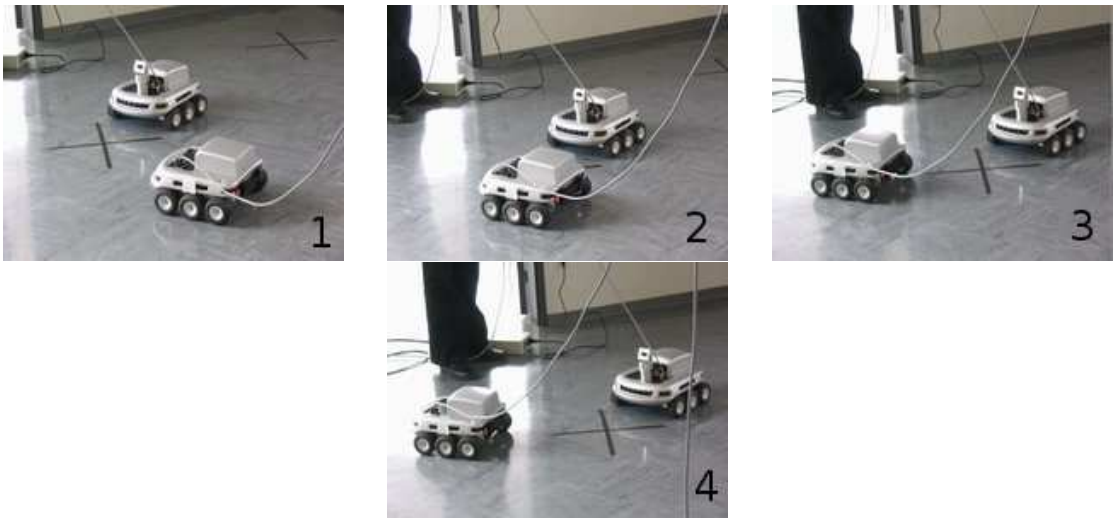

FIG. 6 – Execution de la mission (passage au site D)

$$AR_i(s_i^t) = (R_i(t_i^j|FB_{ij}, LB_{ij}), R_{ij}(F_{ij}|t_i^j), Penalty_{ij}(L_{ij}|t_i^j))$$

telle que :
$R_i(t_i^j|FB_{ij}, LB_{ij})$ est la récompense immédiate que le robot a lorsqu'il réalise la tâche t_i^j étant donné l'état des emnsembles FB_{ij}, LB_{ij}.
$R_{ij}(F_{ij}|t_i^j)$ est la récompense collective lors de la réalisation t_i^j sur les robots réalisant les âches de F_{ij}.
Enfin, $Penalty_{ij}(L_{ij}|t_i^j)$ est le coût occasioné par la réalisation de la tâche t_i^j sur les robots réalisant les tâches de L_{ij}. L'équation de Bellman dans ce contexte est :

$$V_i(s_i^t) = AR_i(s_i^t) + \max_{a_i^k} \sum P_i(s_i^{t+1}, a, s_i^t).V_i(s_i^{t+1}) \tag{6}$$

La résolution de cette équation où les fonctions V_i et AR_i sont des vecteurs consiste à redéfinir un autre opérateur $\max$ qui permet de préférer un vecteur de valeur sur un autre. Une première version de ce travail consistà utiliser un ordre lexicographique permettant ainsi de tenir compte de tous les critères de décisions.

6.2 Allocation dynamique de tâches

Dans notre contexte, les robots évoluent dans un environnement en vue de réaliser des tâches. Ces tâches qui, en général, sont situées dans l'espace vers lesquelles les robots doivent se diriger. La satisfaction individuelle d'un robot telle qu'elle est utilisée dans la section précédente est de se diriger vers une zone où les tâches relevant de ses compétences (soucourir des blessés, éteindre les feux, ect) à réaliser sont nombreuses. Pour sela, nous utilisons des grilles pondérées qui permettent d'associer un poids à une zone selon l'importance des tâches à réaliser 7. Chaque agent choisit les tâches qui maximisent son espérance de gain [Hanna and Mouaddib2002] et lorsqu'un conflit sur une tâche existe, le robot dont le coût occasionné de la tâche est le plus petit est celui qui prend la tâche. Ce coût occasionné est calculé par :

$$OC(t_i^j) = V(s^t) - V(s^{t+1})$$

où $V(s^t)$ est la gain espéré actuel sans la tâche t_i^j est $V(s^{t+1})$ est le gain espéré avec la prise de la tâche $_i^j$. Ce coût est utilisé pour résoudre les conflits.

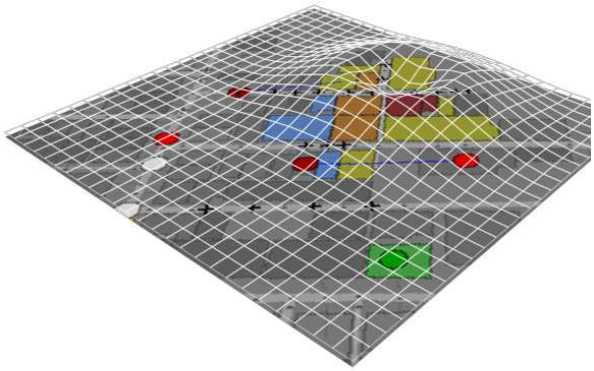

FIG. 7 – Exemple de grille pondérée

7 Architecture Globale

L'architecture de tel syst'eme consiste en trois couches :
– *Allocation de tâches* : Après perception du monde, les robots se construisent des grilles pondérées qui leur permettent de localiser les zones les plus riches en tâches qu'ils se répartissent. Ce processus se répètent jusuq'à

ce que toutes les tâches ou les plus importantes soient allouées. Toutes les tâches peuvent ne pas être allouées à cause des ressources limitées des robots.

– *Coordination Spatiale* : lorsque l'allocation des tâches est effectuées, les robots doivent se mouvoir et choisir l'ordre de réalisation des tâches en tenant compte des contraintes entre les tâches et la satisfaction globale des robots en utilisant l'approche DEC-MDP multicritères.
– *Contrôle de réalisation* : Les deux premières étapes permettent de construire les graphes de tâches et l'allocation telles qu'elles sont utilisées dans le mod'eles OC-DEC-MDP. Ensuite, OC-DEC-MDP contrôle la réalisation et l'évolution en calculant des politiques pour les différents robots.

8 Conclusion

Nous avons proposé une approche basée sur les Processus Décisionnels de Markov Décentralisés et permettant de résoudre des problèmes de contrôle décentralisé dans des systèmes multi-agents coopératifs. Ce modèle, OC-DEC-MDP, permet de traiter des problèmes de grande taille (plus de 200 tâches) et de tenir compte de contraintes temporelles et de précédence entre les tâches. De plus, cette approche ne nécessite pas de communication entre les agents lors de l'exécution, elle peut donc être utilisée dans des systèmes où la communication est impossible ou très coûteuse. Afin que les agents agissent de manière coopérative, nous utilisons la notion de coût occasionné. Elle permet à chaque agent de déterminer une politique locale tenant compte de sa propre utilité mais également de l'impact de ses décisions sur les autres. Le calcul des politiques des agents est réalisé par un algorithme polynomial en espace d'états. Les expérimentations ont montré que la politique obtenue donne de meilleurs résultats que des approches heuristiques et laissent penser que cette politique est très proche de la politique optimale.

Nous avons aussi décrits des extensions possibles de ce modèles pour une meilleure coordination spatiale pour la navigation et l'allocation de tâches. L'architecture globale est définie qu'on utilisera dans plusieurs scénarios en cours d'élaboration pour de flottes de robots.

Références

[Becker *et al.*2003] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Transition-independent decentralized markov decision processes. In *AAMAS*, 2003.

[Becker *et al.*2004] R. Becker, V. Lesser, and S. Zilberstein. Decentralized Markov Decision Processes with Event-Driven Interactions. In *The Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, volume 1, pages 302–309, NYC, 2004. IEEE Computer Society.

[Bernstein *et al.*2000] D. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of mdps. In *UAI*, 2000.

[Goldman and Zilberstein2003] C. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multiagent systems. In *AAMAS*, 2003.

[Hanna and Mouaddib2002] H. Hanna and AI Mouaddib. Task selection as decision making in multiagent system. In *AAMAS*, pages 616–623, 2002.

[http ://www.rescuesystem.org/robocuprescue/] http ://www.rescuesystem.org/robocuprescue/.

[Mouaddib2004] AI Mouaddib. Multi-criteria path planning. In *IEEE International Conference on Robotic and Automaton*, pages 2814–1819, 2004.

[Peshkin *et al.*2000] L. Peshkin, K.E. Kim, N. Meuleu, and L.P. Kaelbling. Learning to cooperate via policy search. In *UAI*, pages 489–496, 2000.

[Pynadath and Tambe2002] D. Pynadath and M. Tambe. The communicative multiagent team decision problem : Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, pages 389–423, 2002.

[Yokoo *et al.*2003] M. Yokoo, R. Nair, S. Marsella, M. Tambe, and D.V. Pynadath. Taming decentralized pomdps : Towards efficient policy computation for multiagent settings. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2003.