Contribution des méthodes d'apprentissage à la distribution de tâches dans un cluster robotique

Paul Gautier

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

Université Bretagne Sud

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE BRETAGNE SUD
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Paul GAUTIER

Contribution des méthodes d'apprentissage à la distribution de tâches dans un cluster robotique.

Thèse présentée et soutenue à Lorient, le 27 avril 2021
Unité de recherche : UMR6285 – Lab STICC
Thèse N° : 591

Rapporteurs avant soutenance :

Cécile BELLEUDY — Maitre de conférences, HDR, LEAT, Université Côte d'Azur
François BERRY — Professeur, Institut Pascal, Université Clermont Auvergne

Composition du Jury :

Président :	Cédric BUCHE	Professeur, IRL CNRS CROSSING
Examinateurs :	Cécile BELLEUDY	Maitre de conférences, HDR, LEAT, Université Côte d'Azur
	François BERRY	Professeur, Institut Pascal, Université Clermont Auvergne
	Cédric BUCHE	Professeur, IRL CNRS CROSSING
	Maher JRIDI	Maitre de conférences, HDR, ISEN
Dir. de thèse :	Jean-Philippe DIGUET	Directeur de recherche, CNRS
Co-dir. de thèse :	Johann LAURENT	Maitre de conférences, HDR, UBS

Remerciements

Je remercie Jean-Philippe Diguet d'avoir permis à ce projet de voir le jour ainsi que pour m'avoir fait confiance durant ces quatre années.

Je remercie Johann Laurent d'avoir accepté de rejoindre l'équipe en cours de projet et d'avoir permis que ce dernier connaisse une fin heureuse.

Je tiens à remercier Cécile Belleudy et François Berry pour avoir accepté d'être rapporteurs de ces travaux.

De même, je remercie également Cédric Buche et Maher Jridi pour leur participation à ce jury de thèse.

Je remercie mes collègues et amis et tout particulièrement Erwan Moréac, Yohann Rioual et Julien Mazuet pour leur aide et leurs conseils qui m'ont été précieux.

Pour terminer, je remercie les membres de ma famille pour leur soutien et plus spécialement ma mère, Anne Faivre, pour la patience et la disponibilité dont elle a fait preuve.

SOMMAIRE

INTRODUCTION

Les systèmes multi-robots (MRS) sont des systèmes composés de plusieurs robots coordonnés en vue d'accomplir une mission. Cette multiplication du nombre de robots permet d'améliorer les performances du système en augmentant notamment les capteurs et actionneurs disponibles tout en réduisant les coûts puisqu'il est plus facile de produire plusieurs robots qu'un seul "super" robot. De plus, la nature interchangeable et multiple des robots composant ces systèmes les rend particulièrement robustes aux pannes puisque la perte d'un robot ne compromet pas fondamentalement leur fonctionnement. L'intérêt des concepteurs pour les systèmes multi-robots ne cesse d'augmenter et ils ont été déployés dans de nombreux contextes allant de l'automatisation d'usines aux missions de recherche et sauvetage en passant par la livraison de marchandises comme illustrés en Figure 1 et 2.

FIGURE 1 – Starship : des robots de livraisons autonome déployés en Arizona.

FIGURE 2 – Système multi-robots MSRBOTS déployé dans le cadre de missions de recherche et sauvetage prenant place dans des mines de charbon [59].

Cependant, afin d'être efficace, un système multi-robots doit être capable de coordonner ses membres en distribuant les tâches au sein du système ce qui conduit au problème du

MRTA (multi-robots task allocation). La difficulté du problème dépend de l'architecture du système (centralisé ou décentralisé) et de son homogénéité. Dans le cas de systèmes décentralisés, les approches basées sur le marché font évoluer les robots dans une économie fictive où les tâches et ressources nécessaires à leur complétion deviennent des produits de consommation pouvant être échangés. Quelle que soit la solution utilisée, les systèmes multi-robots déployés tendent à être de plus en plus autonomes.

Ce besoin d'autonomie s'accompagne de nouvelles exigences liées notamment à la sécurité ainsi qu'à la navigation en conditions réelles. En effet, afin de pouvoir évoluer librement un système multi-robots doit être en mesure de percevoir, analyser et comprendre en temps réel l'environnement dans lequel il évolue. Les récentes avancées technologiques dans les domaines de la fusion de données et de la vision par ordinateur (Figure 3) ont permis d'atteindre certaines de ces exigences. L'implantation de ces méthodes a cependant entraîné une forte augmentation du nombre de capteurs présents sur les robots ainsi qu'une complexification des algorithmes de traitements [31]. Dès lors, les ressources calculatoires intrinsèquement limitées des robots deviennent une préoccupation majeure puisqu'elles conditionnent leur capacité à comprendre et interagir avec leur environnement. Or, si de nombreuses solutions existent pour le problème du MRTA, elles ont, bien souvent, abordé le problème sous l'angle de la localisation du robot et non sous celui de ses capacités de calculs.

Cette difficulté liée aux tâches calculatoires a été partiellement résolue avec l'introduction du cloud robotique [17]. Ce mécanisme permet aux robots d'effectuer des calculs intensifs sur des serveurs distants, d'accéder à de larges bases de données, mais aussi de coordonner l'ensemble du système. Le projet RoboEarth offre un bon exemple des possibilités offertes par le cloud avec sa plate-forme open-source de cloud robotique appelée Rapyuta. La Figure 4 illustre l'architecture système de Rapyuta qui permet aux robots d'exécuter leurs tâches calculatoires dans le cloud via des environnements informatiques personnalisables et sécurisés. Si cette solution permet de s'affranchir des capacités limitées des systèmes embarqués, elle s'accompagne de plusieurs contraintes. D'une part, le système multi-robots devient dépendant du cloud. Une connexion fiable et performante devient alors nécessaire et toute rupture de cette dernière entraîne la paralysie du système. Nous assistons à la création d'un point de défaillance unique. D'autre part, le recours au cloud introduit une latence rendant le système moins réactif, et donc, peu compatible avec

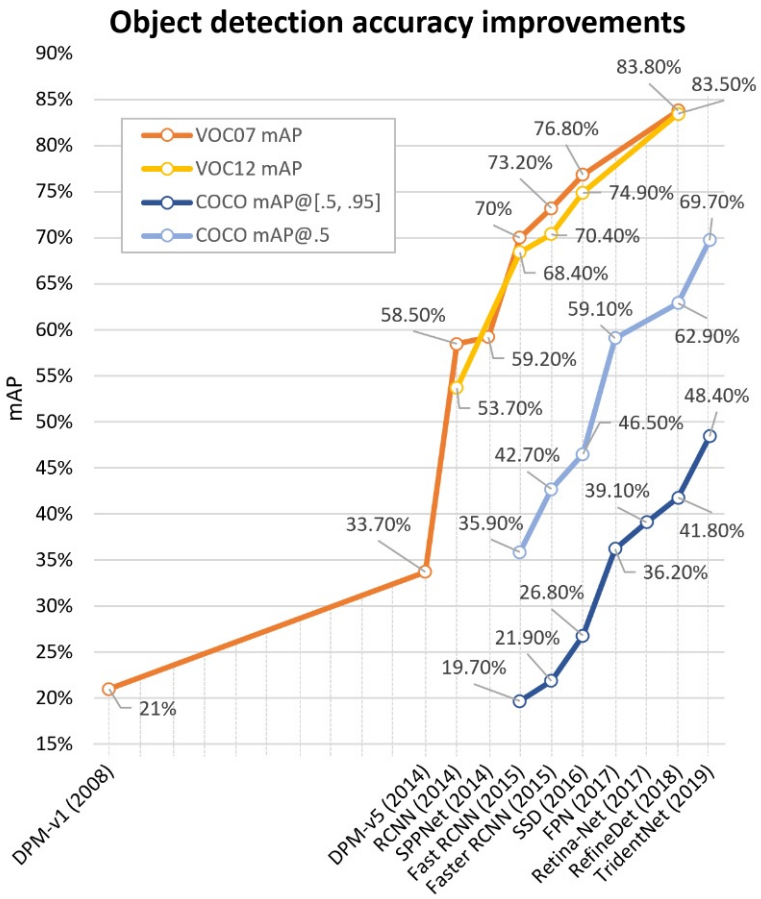

FIGURE 3 – Amélioration de la détection d'objets pour différents algorithmes de vision par ordinateur utilisant des réseaux de neurones [63].

les environnements hautement dynamiques.

Lorsque l'environnement est hostile (connexions vers l'extérieur incertaines et fort degré d'incertitude lié à de brusques changements), le système multi-robots ne doit compter que sur lui même. Devant ce constat, le principe du cluster robotique [26] a été développé pour améliorer les performances calculatoires du système sans recourir à une aide extérieure. Dans un cluster, les ressources calculatoires non utilisées sont mutualisées, comme illustré en Figure 5, afin d'être allouées à la parallélisation de tâches fortement calculatoires. Cette solution permet de maximiser l'emploi des ressources par le système et n'introduit aucune dépendance externe. Son utilisation s'avère cependant complexe car contrairement au cloud robotique, les ressources disponibles s'avèrent limitées et fluctuantes. Dès lors, leur allocation nécessitent un processus dynamique et adaptatif capable de cerner les besoins

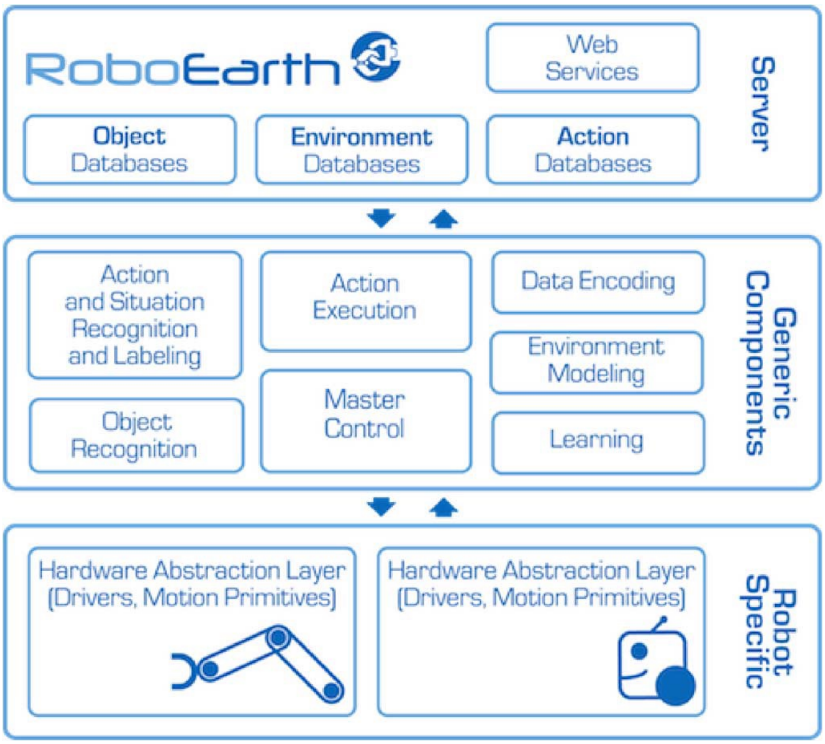

FIGURE 4 – L'architecture du system RoboEarth systems permet aux robots de partager leurs connaissances et d'apprendre les uns des autres [17].

changeants du système.

Or, ce dynamisme pourrait être appréhender par des solutions basées sur l'apprentissage par renforcement. Ce type d'apprentissage automatique est devenu populaire en 2013 lorsque la machine a battu l'Homme sur les jeux Atari 2600, tels que ceux présentés en Figure 6, au moyen de l'algorithme de *Q-learning*. En apprenant, le système parvient à appréhender l'incertitude et à s'adapter à un contexte dynamique. Son emploi comme mécanisme d'allocation au sein d'un cluster robotique pourrait permettre au système d'adapter sa gestion des ressources en fonction du contexte et des variations imposées par l'environnement.

Dans ces travaux de thèse, nous explorons le problème de l'allocation de tâches calculatoires au sein d'un système multi-robots évoluant dans un contexte opératoire dynamique et incertain. Par conséquent, son architecture est nécessairement décentralisée et il ne peut utiliser que ses propres ressources. Nous proposons donc des solutions basées sur le concept du cluster robotique, une méthode encore peu explorée.

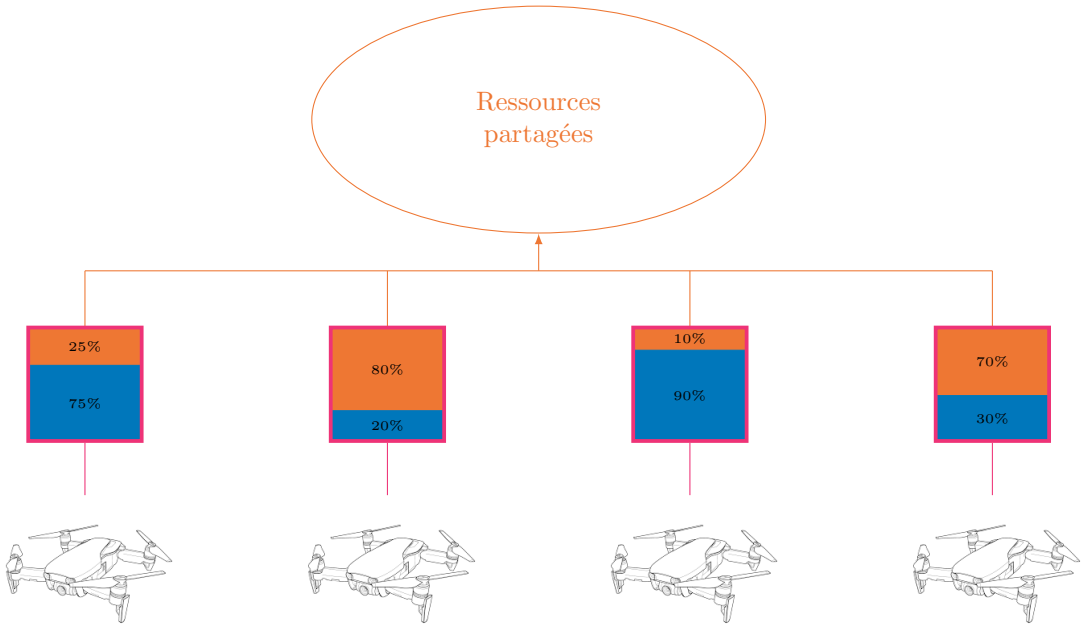

FIGURE 5 – Dans un cluster robotique les ressources de calcul (en magenta) des robots sont divisées en deux parties. La première partie (en bleu) est constituée des ressources utilisées actuellement par le robot afin d'effectuer ses tâches courantes. La seconde partie (en orange) représente les ressources libres du robot. Ces dernières sont mises à disposition du cluster afin d'être partagées [26].

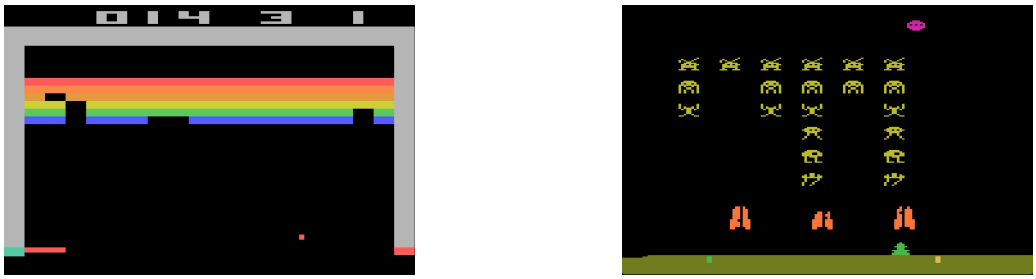

FIGURE 6 – Deux des jeux où la machine est désormais plus forte que l'être humain : à gauche Breakout et à droite Space invader [33].

Au vu du contexte incertain, nous pensons que les méthodes basées sur l'apprentissage par renforcement pourraient offrir le dynamisme nécessaire à l'accomplissement de certaines missions comme celle de recherche et sauvetage. Leur utilisation n'est pas sans limite puisque d'une part, le caractère décentralisé du système rend difficile les échanges d'expériences entre les robots et d'autre part, le problème des espaces d'actions à fortes dimensions n'est toujours pas résolu. L'objectif de cette thèse est d'évaluer les apports

de l'apprentissage par renforcement dans la gestion d'un cluster robotique en tant que substitut aux méthodes classiques ou bien en les épaulant.

Contributions de la thèse

1. *Définition d'une variante du problème du MRTA appelée MRpTA* : de nombreuses solutions ont été proposées pour résoudre le problème MRTA. Cependant, ces traitements ont presque toujours posé la localisation des robots ou leur hétérogénéité comme principale contrainte. Devant l'accroissement du nombre de tâches calculatoires, nous estimons que ces dernières deviennent une contrainte majeure et proposons donc un nouveau paradigme au problème du MRTA.
2. *Proposition d'utilisation d'une méthode d'apprentissage par renforcement comme substitut à une approche basée sur le marché* : les approches basées sur le marché sont couramment utilisées comme méthode de résolution pour les problèmes de MRTA dans le cadre de systèmes multi-robots décentralisés. Cette solution, efficace et peu coûteuse, nécessite cependant d'être capable de valoriser les différentes ressources et tâches. Or, dans un contexte dynamique, l'importance de ces ressources et tâches fluctue, rendant toutes valorisations statiques imprécises. Devant ce constat, nous proposons une autre approche basée sur le *deep Q-learning* qui s'affranchit de tout système de valorisations. Nous comparons ces deux approches dans un contexte de MRpTA où un cluster robotique doit exécuter un maximum de tâches tout en respectant un système de priorités.
3. *Proposition d'utilisation de méthodes d'apprentissage pour optimiser les performances d'un cluster robotique déployé dans un contexte dynamique* : le processus décisionnel d'un cluster robotique doit être capable de s'adapter au contexte dynamique et incertain des missions de recherche et sauvetage. Si les approches classiques offrent un cadre de résolution satisfaisant, elles s'avèrent limitées par leur nature statique. Nous proposons d'utiliser de l'apprentissage par renforcement lors de la conduite d'enchères doubles afin d'optimiser la distribution de tâches calculatoires au sein d'un cluster robotique et de rendre le système adaptatif.
4. *Investigation des méthodes de résolutions utilisant le deep Q-learning pour résoudre des problèmes aux espaces d'actions discrets à fortes dimensions* : Si l'utilisation de réseaux de neurones comme estimateur a permis d'appliquer avec succès l'algorithme de *Q-learning* à des problèmes aux espaces d'états à fortes dimensions, les difficultés

liées aux problèmes possédant des espaces d'actions à fortes dimensions demeurent peu explorées et non résolues. Nous avons comparé les deux approches les plus prometteuses à savoir l'approche par branches et l'approche multi-agents.

Plan de la thèse

Cette thèse est composée de cinq autres chapitres ordonnés de la façon suivante :

Chapitre 1 : contexte théorique : cette partie contient les connaissances nécessaires à la compréhension de l'apprentissage par renforcement. Nous y introduisons les concepts clés ainsi que deux approches de résolution. Ce chapitre se termine sur une introduction aux réseaux de neurones puisque ces derniers sont désormais couramment utilisés comme estimateur en apprentissage par renforcement.

Chapitre 2 : état de l'art : cette seconde partie offre un aperçu de l'état l'art relatif à ces travaux de thèses et est divisée en deux sections. La première expose les avancées du *Q-learning*, et plus particulièrement du *deep Q-learning*, ainsi que son application pour des problèmes aux espaces d'état discret à fortes dimensions. La seconde section traite de l'allocation de tâches dans les systèmes multi-robots et du traitement des tâches calculatoires en robotique.

Chapitre 3 : allocation de tâches calculatoires dans un système multi-robots : cette partie présente une variante du célèbre problème du MRTA basée sur l'allocation de tâches calculatoires. Dans ce contexte, nous comparons deux approches de résolutions : celle basée sur le marché et celle s'appuyant sur le *Deep Q-learning* et évaluons leurs capacités d'adaptation.

Chapitre 4 : optimisation par apprentissage d'un cluster robotique : dans cette dernière partie, nous explorons l'utilisation du *Deep Q-learning* pour améliorer les performances d'un cluster robotique déployé dans un contexte de recherche et sauvetage. Le problème traité possédant un espace d'état discret à fortes dimensions, nous comparons les approches par branches et multi-agents.

Conclusion et perspectives : cette dernière partie résume l'ensemble des travaux effectués dans cette thèse et présente les perspectives ouvertes par ces travaux.

NOTATIONS

s	état
a	action
S_t	l'état à l'instant t
$\mathcal{S}$	l'ensemble des états possibles
A_t	l'action prise à l'instant t
$\mathcal{A}(s)$	l'ensemble des actions possibles dans l'état s
R_t	la récompense reçue à l'instant t qui dépend de S_t, S_{t+1} et A_t
$\mathcal{R}$	l'ensemble des récompenses possibles
π	la politique
π_t	la politique suivie à l'instant t
π_*	la politique optimale
$\pi(a\|s)$	la probabilité de prendre l'action a dans l'état s selon la politique π
G_t	retourne la somme des récompenses attendues après t
$p(s', r\|s, a)$	la probabilité de transitionner de l'état s' avec la récompense r depuis s en choisissant a
$r(s, a)$	la récompense attendue en choisissant a dans l'état s
$v_\pi(s)$	la valeur de l'état s selon la politique π
$v_*(s)$	la valeur de l'état s selon la politique optimale
$q_\pi(s, a)$	la valeur de l'action a dans l'état s selon la politique π
$q_*(s, a)$	la valeur de l'action a dans l'état s selon la politique optimale
$V(s)$	estimation de $v_\pi(s)$ ou $v_*(s)$
$Q(s, a)$	estimation de $q_\pi(s, a)$ ou $q_*(s, a)$
$A(s, a)$	l'*advantage function* mesure l'importance relative de chaque action
δ	l'erreur de différence temporelle (*temporal-difference error*)
y	la cible de différence temporelle (*temporal-difference target*)
γ	le facteur de remise (*discount factor*)
α	le taux d'apprentissage (*learning rate*)
ε	la probabilité de choisir une action aléatoire dans une politique ε-greedy

φ la fonction d'activation (*discount factor*)

w_i le poids lié à l'entrée x_i

w_0 la valeur de seuil

θ l'ensemble des paramètres du réseau de neurones

Chapitre 1

Contexte théorique

Sommaire

Ce chapitre introduit les bases théoriques nécessaires à la compréhension des architectures de *deep Q-learning* décrites dans le chapitre suivant. Il est divisé en deux parties. La première est consacrée à l'apprentissage par renforcement, ses principes, ses notions clés et des approches classiques de résolution. La deuxième, plus succincte, traite des réseaux de neurones qui sont désormais couramment utilisés comme approximateur en apprentissage par renforcement. Le lecteur déjà familiarisé avec ces principes peut se rendre directement au chapitre suivant.

1.1 Apprentissage par renforcement

1.1.1 Principes

Présentation

L'apprentissage par renforcement forme une sous-catégorie de l'apprentissage automatique qui vise à conférer aux systèmes informatiques la capacité d'apprendre à partir de données. Il consiste à la fois en un problème et une classe de méthodes de résolution pour ce problème.

Le problème de l'apprentissage par renforcement nécessite de lier chaque état à une action afin de maximiser un signal de récompense dans un système fermé où chaque action influence au minimum l'état suivant. De plus, ne possédant aucune connaissance *a priori* sur les actions à effectuer, l'agent apprenant se doit de découvrir par lui-même quelles actions sont les plus profitables. Pour cela, l'agent doit être capable de percevoir son environnement, d'agir sur ce dernier et de suivre un objectif en rapport avec l'état de son environnement. En résumé, un problème d'apprentissage par renforcement possède les trois caractéristiques suivantes : circuit fermé, dépourvu d'instruction concernant le choix des actions et les conséquences des actions doivent se répercuter dans le temps.

Toute méthode capable de résoudre ce type de problème peut être considérée comme une méthode d'apprentissage par renforcement. Elle diffère de l'apprentissage supervisé qui, en apprenant à partir d'un jeu d'entraînement, cherche à généraliser sa réponse à des situations non présentes dans le jeu d'entraînement. Ce type de méthode est incompatible avec l'apprentissage par interactions. En effet, il s'avère impossible de concevoir un jeu d'entraînement à la fois correct et représentatif de tous les exemples de situations

pour lesquelles l'agent doit agir. L'apprentissage par renforcement se démarque aussi de l'apprentissage non supervisé. En effet, l'apprentissage non supervisé cherche à trouver des structures cachées reliant des données non labellisées alors que l'apprentissage par renforcement tente de maximiser son signal de récompense. Une dernière spécificité de l'apprentissage par renforcement réside dans le compromis à trouver entre exploration et exploitation. En effet, l'agent se doit d'exploiter ses connaissances pour maximiser sa récompense, mais il doit aussi explorer ses possibilités pour enrichir ses connaissances et sélectionner de meilleures actions en fonction de l'état de l'environnement.

Caractéristiques

Outre l'agent apprenant et son environnement, un système d'apprentissage par renforcement est constitué de quatre éléments clés :

1. Une politique définit le comportement de l'agent apprenant à un instant donné. Elle fait correspondre les actions à entreprendre aux états ressentis. Généralement stochastique, elle constitue l'élément principal de l'apprentissage par renforcement d'un agent puisqu'elle suffit à déterminer son comportement.
2. Le signal de récompense fixe l'objectif du système. En effet, à chaque pas de temps, l'agent apprenant reçoit, de la part de l'environnement, une valeur, sa récompense qu'il cherchera à maximiser dans la durée. Cette dernière représente ce qui est bon ou mauvais pour lui. Elle dépend uniquement de l'état courant de l'environnement et de l'action sélectionnée par l'agent. L'agent ne contrôle pas ce procédé, mais peut, par son choix d'actions, altérer l'environnement, et donc, modifier la récompense. Par conséquent, le signal de récompense constitue l'élément central de l'ajustement de la politique. En effet, lorsque la récompense obtenue lui semble trop faible, l'agent modifie sa politique afin de sélectionner une autre action pour cette situation.
3. Les *value functions* définissent ce qui est rentable sur le long terme. Concrètement, une *state value* représente la quantité totale de récompenses que l'agent peut espérer accumuler à partir de cet état. Ces valeurs permettent de maximiser les récompenses, par conséquent, lors d'une prise de décision, l'agent cherche l'action apportant la plus forte *state value* et non la plus forte récompense. Contrairement aux récompenses obtenues directement par l'environnement, les valeurs doivent être (ré)estimées à partir des observations effectuées par l'agent.
4. Un modèle de l'environnement prédit le comportement de l'environnement, et donc,

permet de planifier les prochaines actions en considérant les situations futures sans avoir à les expérimenter. Le recours à un modèle pré-établi n'est pas automatique car il nécessite d'être en mesure de modéliser l'environnement. Il existe donc deux types d'approches, celles *model-based* ayant recours à la planification et leurs opposées dites *model-free* construisant le leur par essais et erreurs.

Dans la prochaine partie, nous définissons mathématiquement le problème de l'apprentissage par renforcement et introduisons plusieurs concepts fondamentaux, tels que les processus de décisions markoviens [1].

1.1.2 Processus de décision markovien fini

En apprentissage par renforcement, l'apprenant et preneur de décisions s'appelle l'agent. Il interagit au moyen d'actions, de façon continue, avec l'extérieur dénommé environnement. L'environnement répond à chacune des actions de l'agent en lui transmettant un nouvel état ainsi qu'une récompense. Une spécification complète d'un environnement définit une tâche, une instance du problème d'apprentissage par renforcement.

Concrètement, pour chaque pas de temps discret $t = 0, 1, 2, ...$, l'agent reçoit une représentation de l'état de l'environnement $S_t \in \mathcal{S}$ où $\mathcal{S}$ représente l'ensemble des états possibles. Il déduit une action $A_t \in \mathcal{A}(S_t)$ où $\mathcal{A}(S_t)$ dénote l'ensemble des actions disponibles dans l'état S_t. Au pas de temps suivant, l'agent reçoit une récompense $R_t \in \mathcal{R} \subset \mathbb{R}$ ainsi que le nouvel état de l'environnement S_{t+1} comme illustré par la Figure 1.1. Le choix de l'action par l'agent s'effectue selon sa politique noté π_t où $\pi_t(a|s)$ correspond à la probabilité $A_t = a$ si $S_t = s$. La méthode d'apprentissage par renforcement utilisée définit l'ajustement de sa politique, en fonction des expériences vécues, afin de maximiser la somme des récompenses reçues.

Plus précisément, l'agent essaie de maximiser le retour attendu (la somme des récompenses attendues) noté G_t qui correspond à une fonction spécifique de la séquence de récompenses. Dans le cas où l'interaction agent/environnement peut être divisée en épisodes possédant un état final S_T conduisant à la ré-initialisation du système (comme une série de parties d'échecs où chaque partie représente un épisode), le retour attendu est défini par l'équation 1.1.

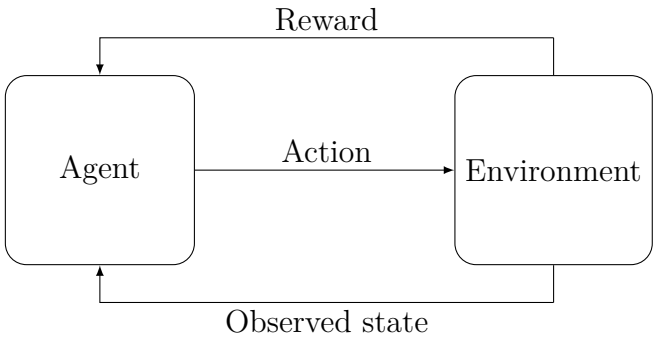

FIGURE 1.1 – L'interaction agent/environnement en apprentissage par renforcement

$$G_t = R_t + R_{t+1} + R_{t+2} + ... + R_T \tag{1.1}$$

Cependant, dans de nombreux cas, l'interaction agent/environnement ne peut être délimitée en épisodes finis et s'apparente à un épisode infini. L'équation 1.1 devient alors problématique puisque ($T = \infty$) ce qui conduit potentiellement à un retour attendu infini. Pour contrer cela, un *discount factor* γ est introduit avec $0 \leq \gamma \leq 1$. L'agent choisit alors A_t de façon à maximiser la somme des récompenses à taux réduit tel que présenté en 1.2. La valeur de γ détermine la vision à long terme de l'agent, de l'insouciant $\gamma = 0$ maximisant uniquement R_t, au prévoyant avec $\gamma = 1$.

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k} \tag{1.2}$$

Un signal d'état détenant toutes les informations nécessaires à la prise de décision satisfait les propriétés de Markov [1]. Par exemple, dans le cas d'une partie d'échecs, un signal comportant la position des pièces satisfait les propriétés de Markov [1]. De nombreuses informations concernant le déroulement de la partie sont perdues, mais elles ne sont pas nécessaires à la prise de décision. Pour résumer, l'état futur dépend uniquement de l'état présent et non des états passés.

$$Pr\{R_t = r, S_{t+1} = s' | S_0, A_0, R_0, ..., S_{t-1}, A_{t-1}, R_{t-1}, S_t, A_t\} \tag{1.3}$$

1. Pour être tout à fait exact, le signal doit aussi comporter les informations relatives aux possibilités de roque, de prise en passant ainsi qu'un compteur du nombre de coups joués depuis la dernière capture ou la dernière avancée de pion. Il s'agit toujours de valeurs de l'état présent et le déroulé conduisant à ces valeurs n'influence pas la prise de décision

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_t = r|S_t, A_t\} \tag{1.4}$$

Une tâche d'apprentissage par renforcement satisfaisant les propriétés de Markov est appelée processus de décision markovien (MDP). De plus, si les espaces d'action et d'état sont finis, il est alors appelé processus de décision markovien fini. Un MDP fini se définit par son état, ses actions possibles et par la dynamique en une étape de l'environnement. Soit un état s, une action a et une récompense r, la probabilité de transition pour chaque paire d'état suivant s' est :

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_t = r|S_t = s, A_t = a\} \tag{1.5}$$

Il est alors possible de calculer la récompense escomptée pour toute paire (état,action) :

$$r(s, a) = \mathbb{E}[R_t|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a) \tag{1.6}$$

ainsi que les probabilités de transition d'états :

$$p(s', s|a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a) \tag{1.7}$$

Les propriétés d'un MDP permettent de définir une politique optimale à l'aide de *value functions*.

Value functions

La valeur d'un état s selon une politique π se note $v_\pi(s)$ et représente le retour escompté à partir de l'état s en suivant la politique π. Pour un MDP, la fonction valeur pour une politique π, $v_\pi(s)$, peut être défini par l'équation suivante :

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k}|S_t = s\right] \tag{1.8}$$

où $\mathbb{E}_\pi[]$ représente la valeur escomptée si l'agent suit la politique π à n'importe quel pas temporel t. De la même façon, nous pouvons définir la fonction valeur action pour une politique π, $q_\pi(s, a)$ représentant la valeur d'une action a pour un état s en suivant la politique π par :

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k}|S_t = s, A_t = a\right] \tag{1.9}$$

Dans le cadre d'un MDP, il est possible de définir une politique optimale notée π_*. Une politique π est dite supérieure à une politique π' si et seulement si $\forall s \in \mathcal{S}, v_\pi(s) \geqslant v_{\pi'}$. Il existe au moins une politique supérieure ou égale à toutes les autres politiques. Toutes les politiques optimales partagent une *state-value function* optimale notée v_* :

$$v_* = \max_\pi v_\pi(s) \tag{1.10}$$

ainsi qu'une *action-value function* optimale notée q_* :

$$q_* = \max_\pi q_\pi(s,a) = \mathbb{E}[R_t + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{1.11}$$

Nous en avons terminé avec les notions importantes liées aux processus de décision markoviens. Nous allons maintenant aborder rapidement deux méthodes de résolution qui ont introduit des concepts fondamentaux utilisés dans l'apprentissage par renforcement.

1.1.3 Programmation dynamique

La programmation dynamique (DP) comporte l'ensemble des algorithmes utilisés pour calculer des politiques optimales étant donné un modèle parfait de l'environnement d'un processus de décision de Markov. Le DP utilise des *value functions* afin d'organiser et structurer la recherche de bonnes politiques. En effet, il est possible de déterminer une politique optimale à partir du moment où nous disposons des *value functions* optimales, v_* ou q_* satisfaisant les équations d'optimalité de Bellman :

$$\begin{aligned} v_(s)* &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_*(s')\right] \end{aligned} \tag{1.12}$$

où :

$$
\begin{aligned}
q_*(s,a) &= \max_a \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma \max a' q_*(s', a') \right]
\end{aligned}
\tag{1.13}
$$

Principes

La première étape consiste à procéder à l'évaluation de la politique (*policy evaluation* ou *prediction problem*) à savoir calculer la *state-value* v_π pour une politique arbitraire π à l'aide de la formule 1.8.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s \right] \\
&= \mathbb{E}_\pi \left[R_t + \gamma v_\pi(S_{t+1}) | S_t = s \right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s'a) \left[r + \gamma v'_\pi \right]
\end{aligned}
\tag{1.14}
$$

Puisque les dynamiques de l'environnement sont connues, l'équation 1.14 est un système constitué de $|\mathcal{S}|$ équations linéaires pouvant être résolu par itérations successives. Soit une séquence de *value function* $v_0, v_1, ...$, avec v_0 choisie arbitrairement mais non terminale, les approximations successives peuvent être obtenues à l'aide des équations de Bellman :

$$
\begin{aligned}
v_{k+1}(s) &= \mathbb{E}[R_t + \gamma v_k(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s') \right]
\end{aligned}
\tag{1.15}
$$

D'après les équation de Bellman, $v_k = v_\pi$ est un point fixe et $\lim_{k \to \infty} v_k = v_\pi$. En étant capable de calculer la *value function* v_π d'une politique π, nous pouvons désormais procéder à la deuxième étape, l'amélioration π. Dans un état s, nous souhaiterions savoir s'il faut changer la politique au profit d'une action déterministe $a \neq \pi(s)$. Pour cela, il suffit de calculer la valeur de l'action a dans l'état s :

$$
q_\pi(s,a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right] \tag{1.16}
$$

Si $q_\pi(s,a) > v_\pi$ alors une politique qui se comporte comme π sauf dans l'état s où elle choisit a s'avère être une meilleure politique. Il est possible d'étendre ce principe à tous les états ainsi qu'à toutes les actions en choisissant pour chaque état s la meilleure action a d'après $q_\pi(s,a)$. Il en résulte une nouvelle politique gloutonne π' tel que :

$$\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right] \tag{1.17}$$

La dernière étape consiste à itérer jusqu'à obtenir la politique idéale en alternant amélioration/évaluation de la politique.

$$\pi_0 \to v_{\pi_0} \to \pi_1 \to v_{\pi_1} \to ... \to \pi_* \to v_*$$

Cependant, la convergence n'apparaît qu'à la limite. Afin d'accélérer le processus, il est possible de tronquer l'itération en itérant qu'une seule fois. C'est à dire en procédant uniquement à une amélioration et une évaluation :

$$v_{k+1} = \max_a \sum_{s',r} p(s',r|s',a)\left[r + \gamma v_k(s')\right] \tag{1.18}$$

Nous venons de décrire les concepts clés du DP qui permettent de mettre en avant deux grands principes de l'apprentissage par renforcement.

Notions clés

En DP, l'itération de la politique consiste en deux actions simultanées : l'évaluation de la politique visant à rendre la state value consistante avec la politique courante et l'amélioration de la politique rendant la politique gloutonne au regard de la *state value* courante. Ce processus d'interaction entre l'amélioration et l'évaluation de la politique est appelé *generalized policy iteration* (GPI) et est illustré en Figure 1.2.

Nous pouvons noter un autre concept clé, le *boostrapping* consistant à mettre à jour des estimations en se basant sur d'autres estimations. Dans le cas du DP, l'estimation des *state values* se basent sur les estimations des *state values* précédentes.

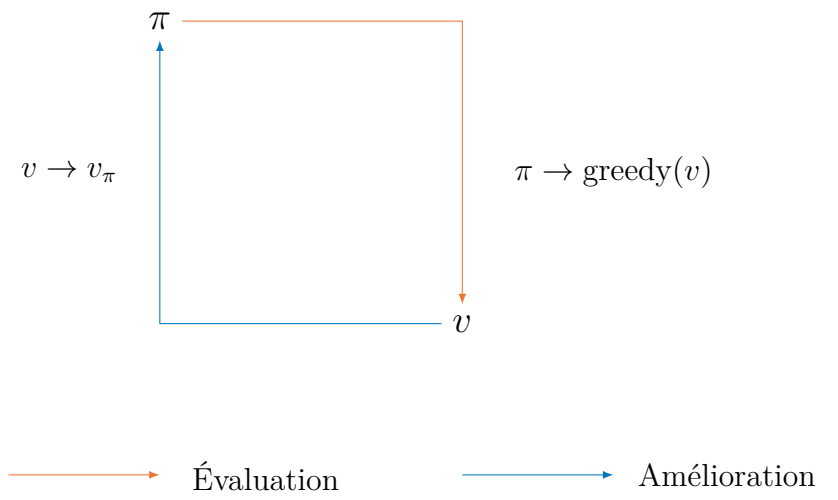

FIGURE 1.2 – GPI : Les *value functions* et *policy functions* interagissent jusqu'à être optimales et donc consistantes l'une envers l'autre.

Après avoir exploré une méthode qui *boostrap* mais exige un modèle précis de l'environnement, nous allons examiner une méthode qui ne nécessite pas de modèle mais ne *boostrap* pas.

1.1.4 Monte Carlo

Dans cette section, nous nous intéressons à la première méthode d'apprentissage qui ne nécessite pas une connaissance complète de l'environnement. En effet, contrairement aux approches de types DP, les méthodes de Monte Carlo apprennent à partir d'expériences à savoir des séquences d'états, actions et récompenses obtenues en interagissant avec l'environnement. Par conséquent, aucune connaissance *a priori* n'est nécessaire.

Principes

La méthode Monte Carlo échantillonne et moyenne la récompense obtenue pour chaque paire état/action plutôt que d'utiliser un modèle pour calculer chaque *state value*. Comme une *state value* correspond au retour attendu, cette moyenne offre une approximation acceptable de cette valeur. Les *action-value functions* sont utilisées car elles permettent d'améliorer la politique sans nécessiter de modèle. L'approche Monte Carlo adopte un procédé GPI sauf que cette fois, la value function est une *action value function* comme décrit en Figure 1.3. Nous retrouvons une itération de la politique classique :

$$\pi_0 \to q_{\pi_0} \to \pi_1 \to q_{\pi_1} \to ... \to \pi_* \to q_*$$

L'amélioration de la politique s'effectue en rendant la politique gloutonne en fonction de l'*action-value function* courante. Pour chaque *action-value function* q, la politique gloutonne correspondante est la suivante :

$$\pi(s) = \arg\max_a q(s, a) \tag{1.19}$$

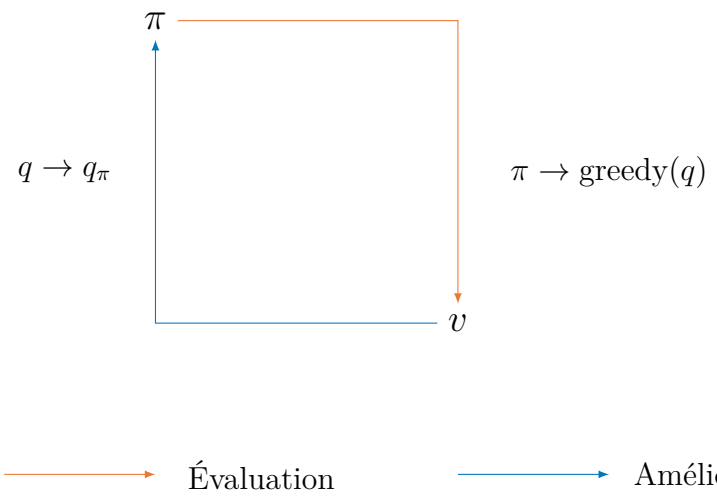

FIGURE 1.3 – GPI : cette fois-ci, l'*action-value function* interagit *policy function* jusqu'à l'optimalité.

Comme le nombre d'épisodes est fini, l'évaluation et l'amélioration de la politique sont effectuées épisode par épisode et peuvent donc être implantées incrémentalement.

Exploration

Il est insuffisant de simplement choisir l'action considérée comme étant la meilleure car aucun retour ne sera obtenu pour les autres actions. Par conséquent, il sera impossible d'apprendre qu'une était meilleure. Deux types de politique existent pour résoudre ce problème :

1. *on-policy* : l'agent s'engage à toujours explorer et cherche à trouver la meilleure politique qui lui permet encore d'explorer.

2. *off-policy* : l'agent explore également, mais apprend une politique optimale déterministe qui peut être indépendante de la politique suivie.

Nous en avons terminé avec le contexte théorique propre à l'apprentissage par renforcement. Nous allons maintenant rapidement exposer les principes des réseaux de neurones qui sont parfois (souvent) utilisés comme approximateur de *l'action-value fonction* en apprentissage par renforcement.

1.2 Réseaux de neurones artificiels

Les réseaux de neurones artificiels sont des modèles statistiques directement inspirés des réseaux de neurones biologiques. Ils sont capables de modéliser et de traiter, en parallèle, des relations non linéaires entre des entrées et des sorties. Un réseau de neurones est constitué d'une association de neurones formels dont le premier modèle fut proposé par Warren McCulloch et Walter Pitts en 1943 [30].

1.2.1 Neurone formel

Un neurone formel est une modélisation mathématique qui imite le fonctionnement d'un neurone biologique. Il possède une ou plusieurs entrées représentant les dendrites et une sortie correspondant à l'axone. Les actions excitatrices et inhibitrices des synapses sont représentées par les poids synaptiques associés aux entrées comme illustré en Figure 1.4. Ces coefficients sont ajustés lors d'une phase dite d'apprentissage. Formellement, un neurone possède les caractéristiques suivantes :

1. n entrées x_i possédant chacune une pondération w_i
2. Une sortie y
3. Une fonction d'activation φ faisant office de seuil d'activation.
4. Une valeur de seuil w_0.

La sortie du neurone est égale à l'image par φ de la somme des entrées pondérées moins la valeur de seuil :

$$y = \varphi\left(\sum_{i=1}^{n} x_i w_i - w_0\right) \tag{1.20}$$

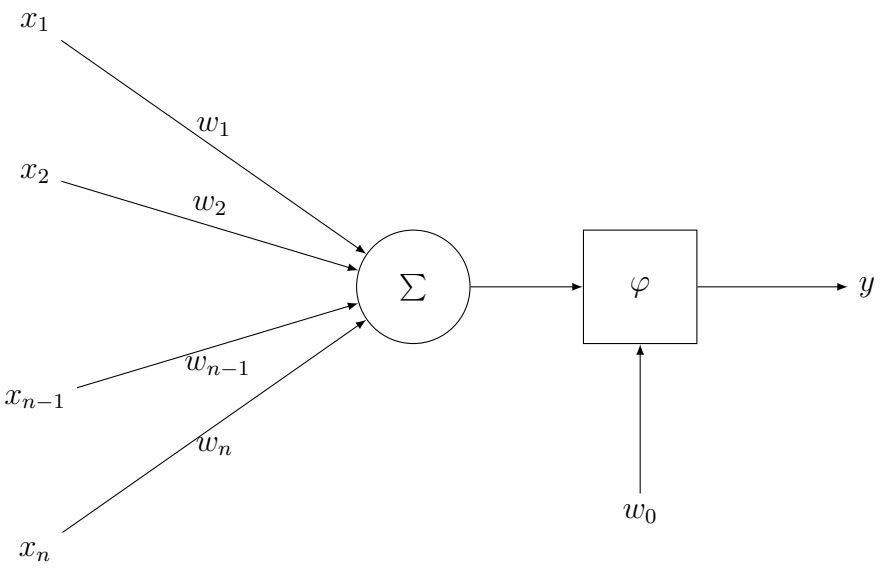

FIGURE 1.4 – Schéma d'un neurone formel : La valeur de sortie du neurone correspond à la somme pondérée des entrées injectée à travers la fonction d'activation après avoir retranché la valeur de seuil.

Il existe de nombreuses fonctions d'activation possédant des caractéristiques particulières, telles que sa non linéarité, sa différentiabilité sur son ensemble de définition, son étendue , sa monotonie ou bien la monotonie de sa dérivée. Les plus connues sont les suivantes :

— Identité : $f(x) = x$

— Heaviside : $f(x) = \begin{cases} 0 & \text{si } x < 0. \\ 1 & \text{si } x \geqslant 0 \end{cases}$

— Sigmoïd : $f(x) = \dfrac{1}{1 + e^{-x}}$

— Unité de Rectification linéaire (ReLU) : $f(x) = \begin{cases} 0 & \text{si } x < 0. \\ x & \text{si } x \geqslant 0 \end{cases}$

L'agrégation de plusieurs neurones en couches permet la formation d'un réseau de neurones.

1.2.2 Réseaux de neurones

Architecture d'un réseau de neurones

Un réseau de neurones est un ensemble d'interconnexions de neurones organisé, le plus souvent, en plusieurs couches comme sur la Figure 1.5. Nous pouvons distinguer trois types de couches :

1. La couche d'entrée recevant les données externes au réseau.
2. La couche de sortie produisant le résultat final.
3. Entre zéro et plusieurs couches cachées réparties entre la couche d'entrée et la couche de sortie et servant à capturer des relations complexes. Chaque couche supplémentaire permet d'extraire des caractéristiques de plus en plus abstraites.

Plusieurs types de connexions sont possibles entre deux couches. Les plus communément utilisées sont les suivantes :

Types	Connexions	Fonction
Fully connected	Chaque neurone est connecté à l'ensemble des neurones de la couche précédente	Extraction de caractéristiques, couche très couteuse
Convolutional	Chaque neurone est connecté à une partie des neurones de la couche précédente	Extraction des caractéristiques d'une image
Pooling	Chaque neurone est connecté à une partie des neurones de la couche précédente	Sous-échantillonnage utilisé pour réduire la taille d'une image
Recurrent	Les neurones sont connectés à des neurones de cette couche ou d'une couche précédente	Adapté aux séries temporelles et signaux acoustiques pour la la reconnaissance de formes

TABLE 1.1 – Les différents types usuels de couches de neurones.

Il existe deux types majeurs de réseaux :

— feed-forward : la propagation s'effectue uniquement vers l'avant.

— récurrents : certains neurones sont connectés à des neurones de leur couche ou d'une couche précédente.

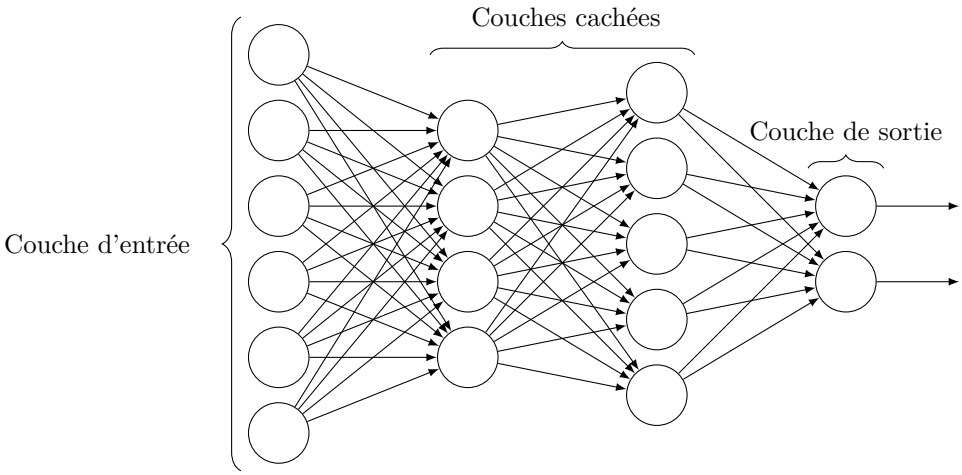

FIGURE 1.5 – Exemple de réseau de neurones artificiels *feed-forward* composé d'une couche d'entrée de sept neurones, de deux couches cachées de respectivement quatre et cinq neurones et d'une couche de sortie de deux neurones. Toutes les couches sont de type *fully connected*.

La force d'un réseau de neurones réside dans sa capacité à apprendre à partir d'exemples sans nécessité de connaissance préalable.

L'apprentissage est l'adaptation du réseau à la réalisation d'une tâche (comme la détection d'objets) à partir d'exemples d'observations. Lors de l'apprentissage, les poids et les seuils du réseau sont ajustés afin d'améliorer la précision de la prediction (inférence). Ce processus s'effectue par minimisation des erreurs observées à l'aide d'une fonction de coût. Ce procédé appelé rétropropagation du gradient permet d'ajuster les poids de connexion afin de compenser les erreurs détectées lors de l'apprentissage. Concrètement, la rétropropagation du gradient calcule le gradient de la fonction de coût associée à un état donné par rapport aux poids [37]. Couramment, la mise à jour des poids est effectuée à l'aide de l'algorithme du gradient stochastique. L'ampleur des corrections apportées aux réseaux dépend du taux d'apprentissage (*learning rate*). L'apprentissage est considéré terminé lorsque les observations supplémentaires ne réduisent plus suffisamment le taux d'erreur observé.

Dans ce chapitre, nous avons abordé les notions clés nécessaires à la compréhension des architectures de type *deep Q-learning* décrites dans le chapitre suivant.

Chapitre 2

État de l'art

Sommaire

Cet état de l'art est divisé en deux parties. La première partie est dévolue à l'apprentissage par renforcement et la seconde traite de la distribution de tâches de calcul au sein d'un système multi-robot (MRS).

2.1 Apprentissage par renforcement

Cette section est entièrement consacrée à l'apprentissage par renforcement, élément central de nos approches de résolution. Le contexte et la problématique abordés dans ces travaux de thèse soulèvent plusieurs défis, principalement liés à la dimensionalité du problème, qui nécessitent l'utilisation de méthodes spécifiques. L'objectif de cette section est de retracer le cheminement scientifique qui a abouti aux développements de ces méthodes.

L'apprentissage par renforcement connait un fort engouement depuis ses succès médiatiques sur les jeux Atari [32], au jeu de Go [44] puis aux échecs [43], au shogi [42] ainsi qu'au jeu vidéo Dota 2 [35]. Les jeux forment un terrain de prédilection pour l'apprentissage par renforcement car ils offrent, de part leur nature, un accès facile à un nombre illimité d'expériences. Cependant, l'apprentissage par renforcement a aussi été appliqué avec succès à de nombreux autres domaines tels que la chimie [62], la robotique [21], l'allocation de ressources [25], l'économie avec les enchères en temps réel [15] et même pour la sélection de recommandations personnalisée en ligne [61]. Il existe de nombreuses méthodes de résolutions en apprentissage par renforcement : *actor-critic*, *policy gradient*, SARSA, *Q-Learning*, etc. Nous nous sommes concentrés sur les méthodes basées sur l'algorithme de Q-learning (deep Q-Learning) qui s'avèrent particulièrement efficaces lorsque les expériences sont difficiles à obtenir.

2.1.1 Temporal-difference

Une des nouvelles idées les plus marquantes dans le domaine de l'apprentissage par renforcement fut sans aucun doute le temporal-difference learning (TD) théorisé par Sutton en 1988 [45]. Cette approche combine le *bootstrapping* du DP et, comme les approches de type Monte-Carlo, la capacité d'apprendre directement depuis des expériences vécues (sans recourir à un modèle de l'environnement).

Temporal-difference learning

Contrairement aux méthodes de type Monte-Carlo, les méthodes de type TD nécessitent uniquement de connaitre la prochaine étape de l'épisode et non l'épisode complet pour incrémenter $V(s)$. La formule de mise à jour d'un état s de l'approche TD la plus simple, le TD(0), qui ne se soucie que du prochain état s' est la suivante :

$$V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{2.1}$$

avec :

— α : le *learning rate*.

— γ : le *discount factor*.

— r : la récompense obtenue dans l'état s

Concrètement, là où l'approche de type Monte-Carlo utilise G_i (G_t dans le chapitre précédent) pour la mise à jour, TD se base sur le cible TD y_i définie dans l'équation 2.2. Comme les approches de type DP, les méthodes TD basent leur mise à jour sur l'estimation de la qualité des états, elles *bootstrap*.

$$y_i^{TD(0)} = r + \gamma V(s') \tag{2.2}$$

Une autre équation importante pour le reste de la section est l'erreur TD correspondant à l'erreur de prédiction de la récompense :

$$\delta_i = r + \gamma V(s') - V(s) \tag{2.3}$$

Les méthodes de type TD s'affranchissent de deux grandes contraintes applicatives :

1. D'une part, contrairement aux méthodes de type DP, elles ne nécessitent pas de modèle de l'environnement.
2. D'autre part, elles dépassent les limitations des approches de type Monte Carlo en étant applicables à des problèmes aux épisodes longs ou continus (sans épisode).

Q-Learning

Une autre avancée marquante dans le domaine de l'apprentissage par renforcement fut l'introduction de l'algorithme de Q-Learning par Watkins en 1989 [55] dont il prouvera la

convergence en 1992 [56] qui sera ensuite généralisée en 1994 par Jaakkola [13] et Tsitsiklis [50]. Cette méthode change le paradigme en considérant des transitions de paires (état, action) en calculant des *Q-values*, là où l'approche TD se limitait aux transitions d'états en apprenant des *state-values*. La formule de mise à jour d'une *Q-value* pour une action a prise dans un état s est la suivante :

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.4}$$

Nous retrouvons la formule d'erreur TD :

$$\delta = r + \gamma \max_{a'} Q(s',a') - Q(s,a) \tag{2.5}$$

Ainsi que celle de cible TD :

$$y_i^Q = r + \gamma \max_{a'} Q(s',a') \tag{2.6}$$

Il s'agit d'une méthode *off-policy* car la fonction Q apprise estime directement l'*action-value function* optimale q_*, et ce, indépendamment de la politique suivie. Cependant, cette dernière souvent de type $\varepsilon - greedy$ détermine quelles paires (état, action) sont sélectionnées et donc mise à jour.

Tabular Q-Learning

Une première approche pratique consiste à stocker les *Q-values* dans une table dont les axes représentent respectivement les états et actions comme illustré en Figure 2.1. Chaque valeur de la table représente une *Q-value* pour une paire (état, action) dont la valeur est mises à jour à l'aide de l'équation 2.4.

Cependant, le recours à une table pour stocker les *Q-values* entraîne deux limitations :

1. D'une part, cette approche est sujette au phénomène de fléau de la dimension. En effet, la taille de la table augmente exponentiellement avec le nombre de paires (état, action) rendant impraticable son parcours et accroissant son empreinte mémoire.
2. D'autre part, l'utilisation d'une table nécessite que les espaces d'états et d'actions soient discrétisés ce qui, à cause de la première limitation, rend inapplicable cette méthode de résolution pour de nombreux problèmes.

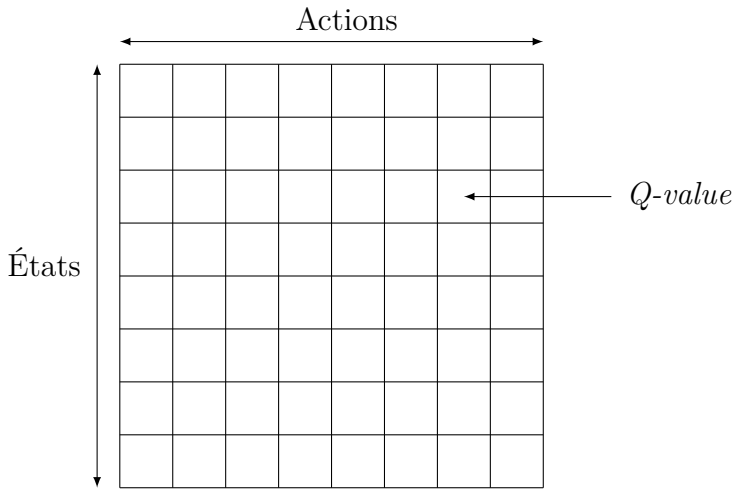

FIGURE 2.1 – Table de Q-Learning : chaque case de la *Q-table* contient une *Q-value* pour une action dans un état.

2.1.2 Deep Q-Learning

Afin de dépasser les limitations induites par l'utilisation d'une table, il est possible d'utiliser un réseau de neurones comme estimateur.

Deep Q-Network

La première utilisation conjointe d'un réseau de neurones avec l'algorithme de Q-Learning remonte aux travaux de Jin [23], mais elle ne concernait qu'un espace d'état réduit. Cette approche n'est devenue populaire qu'avec Minh et al. [33]. Profitant des récentes avancées observées dans l'utilisation d'un réseau de neurones convolutifs profonds, ils ont réussi à traiter un problème d'apprentissage par renforcement possédant un espace d'état à forte dimension (les pixels d'une image).

Concrètement, le *Deep Q-network* a recours à un réseau de neurones de paramètre θ pour approximer la *value function*. Le *Q-Network* est entraîné en minimisant une séquence de fonctions objectif $L_i(\theta_i)$ qui évolue à chaque itération i.

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(y_i^{DQN} - Q(s,a;\theta_i))^2] \tag{2.7}$$

Avec la cible suivante :

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \tag{2.8}$$

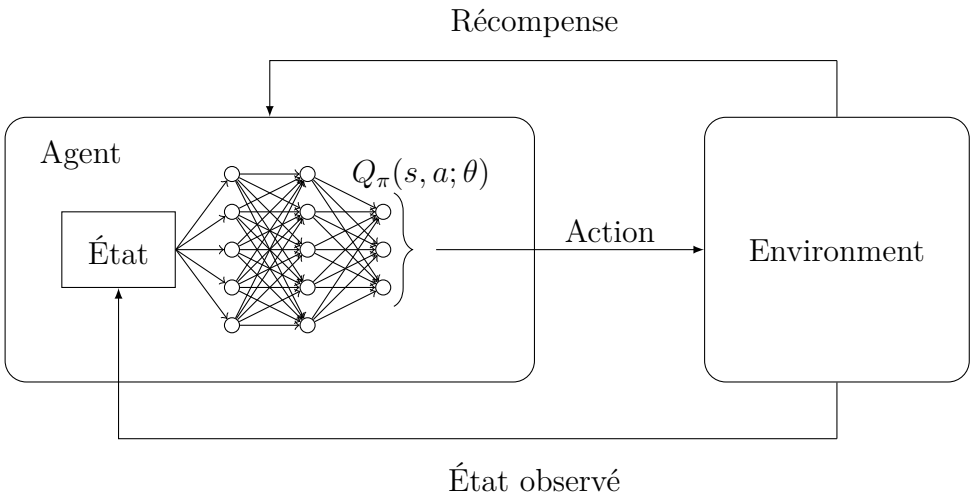

FIGURE 2.2 – Diagramme d'un agent utilisant l'algorithme de *Q-Learning* avec un DQN comme estimateur. Le réseau de neurones reçoit en entrée l'état observé du système par l'agent et prédit, en sortie, les *Q-values* relatives à l'état observé.

Le Q-Network est mis à jour à chaque itération par descente de gradient dont l'équation est la suivante.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a,r,s'} \left[\left(y_i^{DQN} - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right] \tag{2.9}$$

Il s'agit d'une approche *model free* puisque les états et récompenses proviennent de l'environnement. De plus, les états et récompenses sont obtenus à l'aide d'une politique comportementale ($\varepsilon - greedy$) qui diffère de la politique en cours d'apprentissage.

Cette première association de l'algorithme de *Q-Learning* avec un réseau de neurones profond souffre d'instabilité due à deux facteurs.

1. La cible y_i est mouvante.
2. Il existe une forte corrélation entres deux expériences consécutives.

Cette première limitation est résolue en employant un second réseau, le *target network*.

Target network

Nous remarquons que la cible, y_i^{DQN}, utilisée lors de la mise à jour des *Q-values* est générée à l'aide d'un réseau de paramètres θ mis à jour à chaque itération i. Par conséquent, la cible calculée évolue à chaque itération, devenant mouvante et entraînant l'instabilité du réseau. Afin de rendre la cible stationnaire, Minh et al. [32, 33] ont eu recours à un second réseau de paramètres θ^-, le *target network* faisant office de version gelée du réseau originel, l'*online network*. Il est déployé en parallèle et mis à jour toutes les c itérations par clonage des paramètres θ de l'*online network* comme illustré par la Figure 2.3.

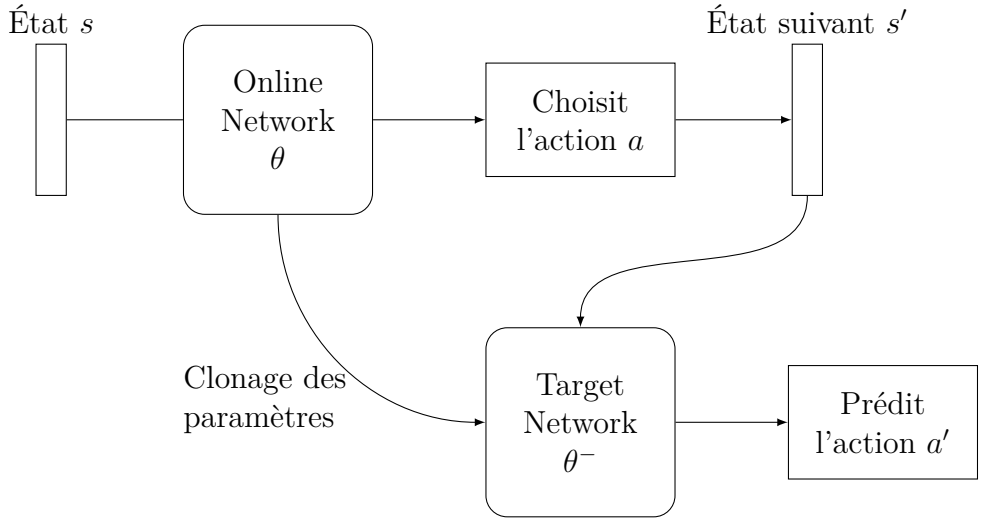

FIGURE 2.3 – Schéma d'un réseau utilisant un *target network* : le calcul de la cible s'effectue selon le choix d'action de l'*online network*, mais avec les paramètres du *target network*

Ce *target network* intervient lors du calcul de la cible la rendant stationnaire :

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{2.10}$$

Cependant, une limitation demeure car le remplacement de la fonction d'estimation des *Q-values* par un réseau de neurones profond conduit à l'instabilité de l'algorithme. En effet, chaque mise à jour d'une valeur Q d'une action entraîne la modification de tous les poids du réseau, et par conséquent, affecte la valeur de chaque action dans les

autres états. Cela impacte fortement la distribution d'échantillonnage. Pour contrer cela, un mécanisme d'*experience replay* peut être mis en place.

Experience replay

Le problème lié à la corrélation d'expériences consécutives fut adressé par Lin [23] puis par Mnih [32, 33] en ajoutant un mécanisme de stockage d'expériences. Pendant l'apprentissage, l'agent stocke dans une mémoire, $\mathcal{D}_{\sqcup} = e_1, e_2, ..., e_t$, les expériences, $e_i = (s_i, a_i, r_i, s_{i+1})$, obtenues à chaque itération i. Ensuite, contrairement au TD où l'expérience courante est utilisée, l'agent s'entraine sur un échantillon d'expériences choisies de façon aléatoire et uniforme dans $\mathcal{D}_{\sqcup}$. La séquence de fonctions objectif devient alors la suivante :

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right] \tag{2.11}$$

Le recours au mécanisme d'*experience replay* améliore considérablement les performances de l'architecture en offrant :

1. Une rupture de la corrélation par réduction de la variance en sélectionnant de façon aléatoire et uniforme les expériences de l'échantillon.
2. Une accélération de la convergence en maximisant l'utilisation des données par l'exploitation d'une même expériences de multiple fois.

Experience replay hiérarchisée

Ce mécanisme d'*experience replay* fut amélioré en 2016 par Schaul [41] qui introduisit le principe de hiérarchisation des expériences. L'objectif recherché était de rejouer plus fréquemment les transitions possédant un potentiel d'apprentissage élevé. Ce potentiel correspond à l'ampleur de leur erreur TD, $|\delta_i|$ qui correspond dans le cas d'une architecture DQN à :

$$\delta_i = r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \tag{2.12}$$

Afin d'atténuer le phénomène de sur-apprentissage dû au manque de diversité induit par cette hiérarchisation, un échantillonnage stochastique est utilisé faisant l'interpolation

entre une hiérarchisation gloutonne et un échantillonnage uniformément aléatoire. La probabilité P d'échantillonner la transition i est alors :

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{2.13}$$

où $p_i > 0$ représente la priorité de la transition i, α définit le degré de hiérarchisation ($\alpha = 0$ correspond au cas uniforme). Concernant la définition de p_i, deux variantes ont été proposées. La première, directe, où $p_i = |\delta_i| + \varepsilon$ introduit un ε qui est une faible valeur positive permettant l'échantillonnage d'expériences d'erreur TD nulle. La seconde, $p_i = \frac{1}{\text{rang}(i)}$, utilise un classement où les transitions sont classées par $|\delta_i|$ décroissantes.

Dans le cas d'une mise à jour stochastique, l'estimation de la valeur attendue nécessite que les mises à jour possèdent la même distribution que celle escomptée. Cependant, en modifiant cette distribution, la hiérarchisation introduit un biais qui altère la solution vers laquelle convergeront les estimations. Ce biais est corrigé par le mécanisme d'échantillonnage des poids importants suivant :

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{2.14}$$

où β représente le degré de compensation et N la taille de la mémoire. Les poids ainsi obtenus peuvent être insérés dans l'algorithme de mise à jour du Q-Learning en remplaçant δ_i par $w_i\delta_i$ (se référer à [24] pour plus de détails).

Si les deux limitations discutées précédemment découlaient de l'utilisation d'un réseau de neurones, un problème lié à la surestimation de l'*action-value* fut découvert au sein de l'algorithme de Q-Learning

Double Q-Learning

Dans l'algorithme de Q-Learning, l'opérateur *max* de l'équation 2.6 utilise les mêmes valeurs pour estimer et sélectionner une action entraînant un phénomène de surestimation des valeurs [49]. Afin de remédier à cela, van Hasselt a proposé une amélioration du Q-learning [11], le Double Q-Learning qui sépare la sélection de l'estimation. Pour cela, deux *values functions* apprennent en parallèle en attribuant aléatoirement chaque expérience à l'une des deux fonctions conduisant à la création de deux jeux de paramètres θ^A et θ^B.

À chaque mise à jour, un des jeux est utilisé pour déterminer la politique gloutonne et l'autre pour calculer sa valeur. Le calcul de la cible y_i^{DQ} devient le suivant :

$$y_i^{DQ} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta^A); \theta^B) \tag{2.15}$$

Double Deep Q-Network

Van Hasselt proposera ensuite une nouvelle architecture le Double Deep Q-Network (DDQN) qui applique le principe du *Double Q-Learning* à une architecture DQN. Cette architecture profite de la présence du *target network* pour tirer parti du Double Q-Learning sans surcoût. En effet, bien que partiellement corrélé à l'*online network*, ce réseau fait office de candidat naturel comme second estimateur sans nécessiter l'ajout d'un nouveau réseau. La politique gloutonne est toujours évaluée par l'*online network*, mais sa valeur est estimée par le *target network*. Il s'agit d'une modification minimale de l'architecture DQN dont la mise à jour reste identique, seul le calcul de la cible y_i^{DDQN} diffère :

$$y_i^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta^-) \tag{2.16}$$

Dueling Network Architecture

Wang et al. [54] ont constaté que dans un grand nombre d'états, il n'est souvent pas nécessaire d'estimer la valeur du choix de chaque action. En effet, certaines actions sont pertinentes uniquement pour des états spécifiques (ex : l'action "évitement d'obstacle" nécessite la présence d'un obstacle). Par contre, pour toutes les approches de type *boostrapping*, connaître les *state-values* pour chaque état est crucial. En conséquence, Wang et al. [54] ont défini l'*advantage function*, $A(s, a)$ qui soustrait la valeur de l'état de la Q *function* pour obtenir la mesure (relative) de l'importance de chaque action :

$$A(s, a) = Q(s, a) - V(s) \tag{2.17}$$

Afin d'intégrer ce principe, Wang et al. [54] ont proposé une nouvelle architecture le Dueling Deep Q-Network (3DQN) séparant les estimations de la *value function* et de l'*advantage function* au moyen de deux branches (streams). Cette architecture est illustrée par la Figure 2.4.

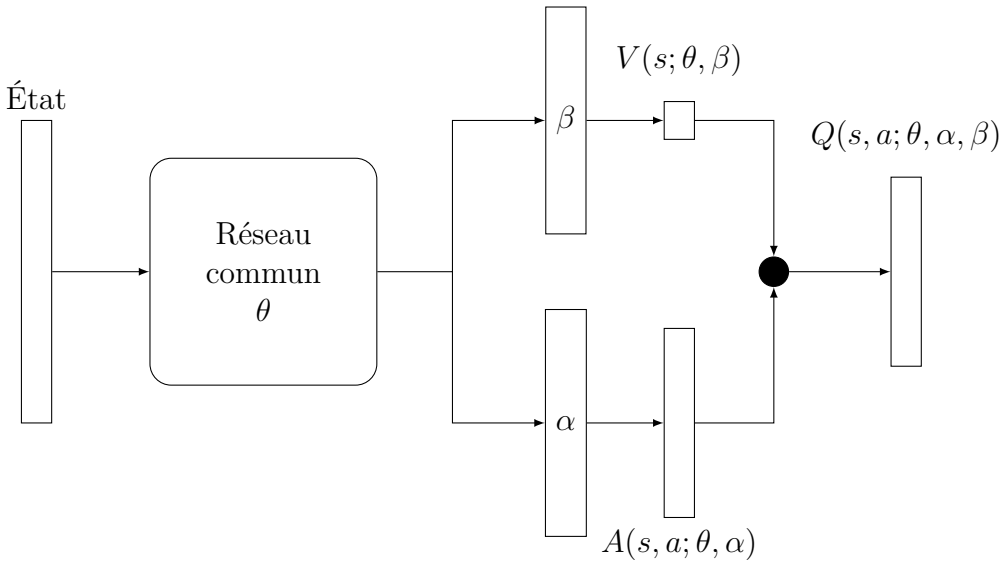

FIGURE 2.4 – Schéma d'une architecture 3DQN : Cette architecture est constituée de quatre éléments distincts. Le premier est un réseau partagé servant à l'extraction des caractéristiques communes. De ce réseau émergent deux branches. La première de paramètre β calcule un unique scalaire, le *state value*. La deuxième de paramètre α produit un vecteur contenant l'*advantage* de chaque action. Le dernier élément est la couche d'agrégation permettant l'estimation des *Q-values*

Les principales caractéristiques de cette architecture sont les suivantes :

— Un réseau partagé de paramètre θ servant à l'extraction des caractéristiques communes aux deux branches. Il s'agit d'un CNN dans le cas de Wang et al.

— Une première branche de paramètre β émergeant du réseau partagé et produisant un scalaire $V(s;\theta,\beta)$.

— Une deuxième branche parallèle de paramètre α engendrant un vecteur $A(s,a;\theta,\alpha)$ de dimension $|\mathcal{A}|$.

— Une couche spéciale réalisant l'agrégation des deux branches pour former $Q(s,a;\theta,\alpha,\beta)$.

Cependant, il est important de noter que $Q(s,a;\theta,\alpha,\beta)$ représente uniquement une estimation paramétrée de la véritable fonction Q. Par conséquent, $V(s;\theta,\beta)$ et $A(s,a;\theta,\alpha)$ ne représentent pas respectivement un bon estimateur de la *state-value function* et une bonne estimation de l'*advantage function*. Plusieurs approches ont été testées pour réaliser l'agrégation. L'approche naïve consistant à réaliser une simple somme des sorties des deux branches 2.18 se heurte à un manque d'identité. En effet, il est impossible d'obtenir

uniquement V ou A pour un Q donné.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) \tag{2.18}$$

Ce problème fut résolu en forçant l'estimateur de l'*advantage function* à ne fournir aucun *advantage* à l'action choisie 2.19. Cette approche fut ensuite améliorée en remplaçant l'opérateur max par une moyenne 2.20.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |\mathcal{A}|} A(s,a';\theta,\alpha)\right) \tag{2.19}$$

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right) \tag{2.20}$$

L'architecture 3DQN permet d'appliquer les mêmes méthodes que les approches de type DQN décrites précédemment, tout en accélérant la convergence lorsque le nombre d'actions possibles augmente. En effet, comme les résultats l'ont attesté, le partage de la valeur $V(s;\theta,\beta)$ entre des actions similaires pour un état s facilite la convergence.

2.1.3 Espace d'actions à forte dimension

Présentation du problème

L'utilisation de réseaux de neurones comme approximateur pour les méthodes d'apprentissage par renforcement a permis de traiter des problèmes aux espaces d'état à forte dimension par modélisation de données hiérarchiques complexes et de caractéristiques. Cependant, les méthodes de type DQN souffrent des mêmes limitations que les approches par table lorsqu'il s'agit de traiter des espaces d'action à forte dimension. En effet, le nombre d'actions devant être représentées augmente exponentiellement avec l'accroissement du nombre de dimensions de l'espace d'action. Si nous considérons un environement possédant un espace d'action à N dimensions composées chacune de n_d sous-actions discrètes pour chaque dimension d, alors le nombre total d'actions devant être considérées est $\prod_{d=1}^{N} n_d$. Ce nombre conséquent d'actions rend rapidement ces problèmes intraitables par des algorithmes à action discrète car l'exploration devient alors inefficace[22]. Deux approches ont été proposées pour appliquer l'algorithme de *Q-Learning* à des problèmes aux espaces d'action à forte dimension.

Dans les deux sous-sections suivantes, nous considérerons un espace d'action à N dimensions où chaque dimension d possède $|\mathcal{A}_d| = n$ discrètes sous-actions.

Approche multi-agents

Une approche intuitive consiste à diviser notre agent en plusieurs sous-agents traitant chacun une dimension du problème. La combinaison d'actions des sous-agents forme alors la réponse de l'agent.

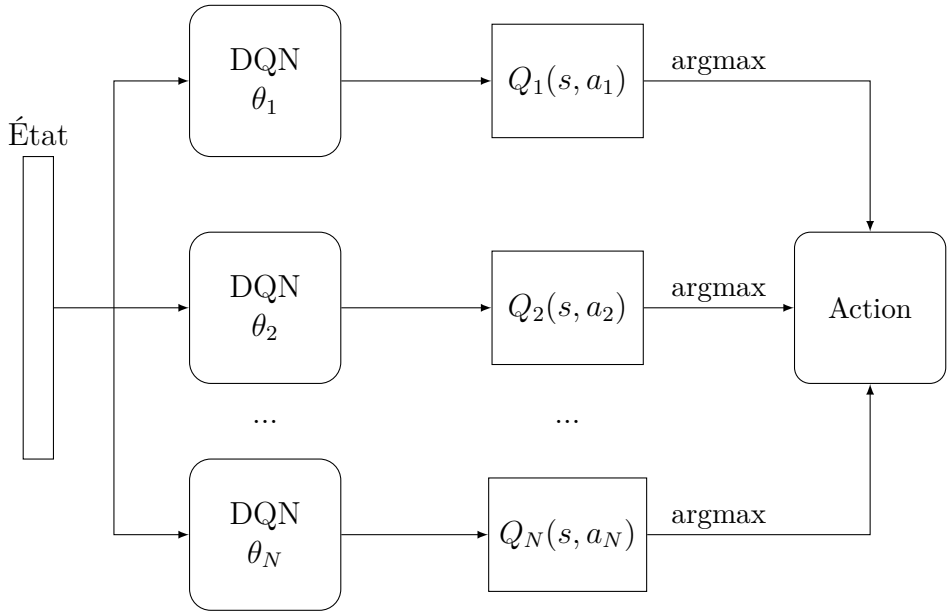

FIGURE 2.5 – Schéma de N DQN indépendants (IDQN) disposés en parallèle où chaque agent (DQN) prédit indépendamment des autres, les *Q-values* relatives à sa dimension. La sélection des meilleurs *Q-values*, pour chaque dimension à l'aide de l'opérateur argmax, forme l'action finale.

Cette méthode soulève deux limitations. D'une part, elle multiplie par N le nombre de réseaux devant être déployés entrainant un surcoût calculatoire ainsi qu'une augmentation de l'empreinte mémoire. D'autre part, comme attesté par [28], l'apprentissage effectué par des réseaux indépendants soulève des problèmes de convergence. Cette limitation fut partiellement adressée par les auteurs de [46] qui, en introduisant un système de récompense par équipe, proposèrent une approche de résolution totalement coopérative. Bien qu'appliquée avec succès à un problème à deux agents (deux dimensions), cette

solution se heurte, en principe, à un problème de convergence pour un espace d'action à plus forte dimension [28].

Branching Dueling Double Deep Q-Network

La deuxième approche proposée par Tavakoli [48] en 2019 constitue une adaptation de l'architecture 3DQN décrite précédemment pour les espaces d'action à forte dimension. L'architecture Branching Dueling Double Deep Q-Network (BDQ) illustrée en Figure 2.6 est composée d'un réseau partagé servant à l'extraction des caractéristiques communes, d'une branche par dimension d'action et d'une unique branche pour le calcul de l'estimation de la *state-value function*. Une agrégation est réalisée entre chaque branche action-dimension et la branche d'estimation afin d'obtenir chaque sous-action. L'action finale est ensuite reconstituée par combinaison des sous-actions sélectionnées.

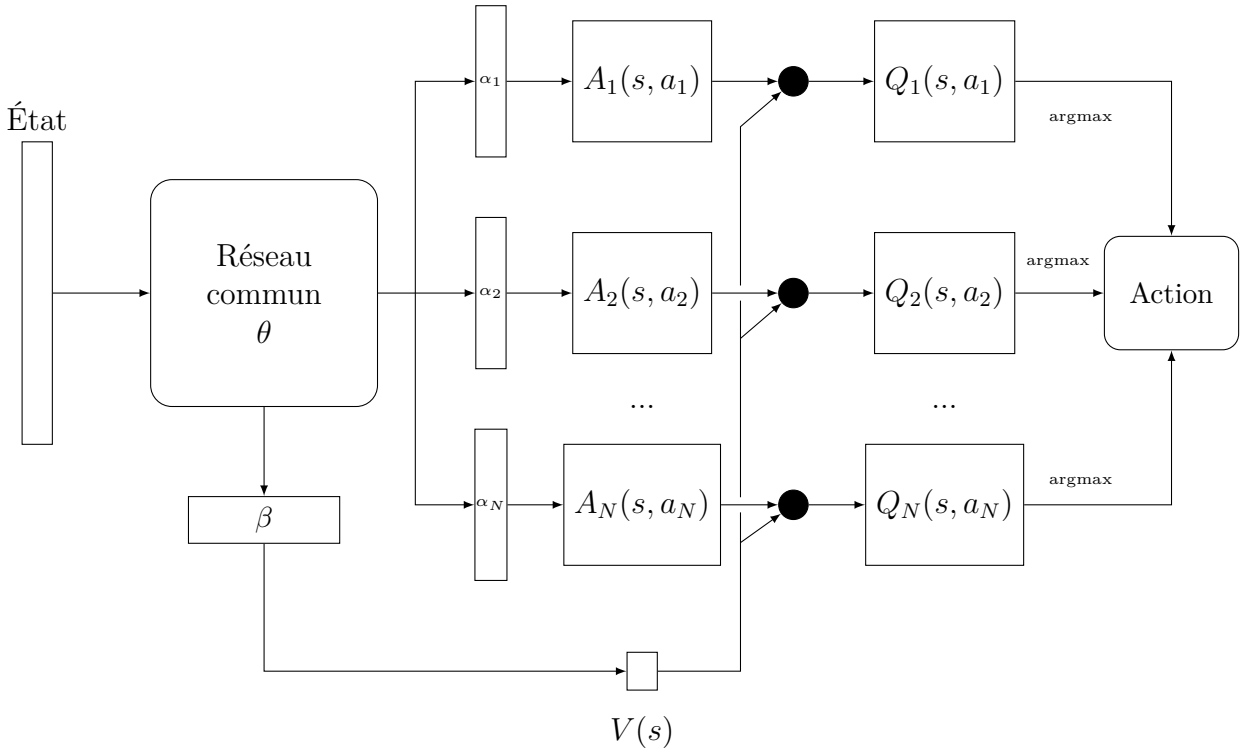

FIGURE 2.6 – Architecture BDQ : nous retrouvons le réseau partagé du 3DQN duquel $N+1$ branches émergent. N branches calculent chacune les *action-advantage* d'une dimension de l'espace d'action pendant qu'une branche estime la *state-value* commune à tout le réseau. Une agrégation est ensuite réalisée entre chaque branche *action-advantage* et la branche de *state-value* afin de produire les *Q-values* de chaque dimension de l'espace d'action. L'action finale est ensuite obtenue par combinaison des meilleures sous-actions.

Plusieurs modules d'agrégation ont été testés adpatant les propositions de Wang et al. [54] à l'approche par branches :

$$Q_d(s, a_d) = V(s) + A_d(s, a_d) \tag{2.21}$$

$$Q_d(s, a_d) = V(s) + (A_d(s, a_d) - \max_{a'_d \in \mathcal{A}_d} A_d(s, a'_d)) \tag{2.22}$$

Comme l'approche de type 3DQN l'utilisation d'une moyenne procure les meilleurs performances :

$$Q_d(s, a_d) = V(s) + (A_d(s, a_d) - \frac{1}{n} \sum_{a'_d \in \mathcal{A}_d} A_d(s, a'_d)) \tag{2.23}$$

À l'instar de l'agrégation, plusieurs méthodes ont été testées pour générer la cible TD lors de la mise à jour du réseau. Une première approche, similaire au DDQN, où une cible est calculée séparément pour chaque branche :

$$y_d^{BDQ} = r + \gamma Q_d^-(s', \arg\max_{a'_d \in \mathbb{A}_d} Q_d(s', a'_d)) \tag{2.24}$$

En fixant une cible d'apprentissage globale pour l'ensemble des dimensions, la seconde méthode offre de meilleurs performances :

$$y^{BDQ} = r + \gamma \max_d Q_d^-(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d(s', a'_d)) \tag{2.25}$$

Elle peut être perfectionnée en remplaçant l'opérateur max par une moyenne :

$$y^{BDQ} = r + \gamma \frac{1}{N} \sum_d Q_d^-(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d(s', a'_d)) \tag{2.26}$$

Cette architecture BDQ s'est révélée concluante pour des espaces actions à forte dimension ($n = 33$ dimensions et 6.5×10^{25} actions) sans pour autant nécessiter le déploiement de davantage de ressources calculatoires supplémentaires.

2.2 Distribution et parallélisation de tâches

Nous quittons le domaine de l'apprentissage par renforcement pour nous intéresser à la distribution de tâches calculatoires au sein d'un système multi-robots.

2.2.1 Multi-Robot Task Allocation

La question de l'allocation des tâches dans un MRS s'avère cruciale pour la coordination du système et forme depuis plusieurs décennies un problème formalisé appelé multi-robot task allocation (MRTA).

Présentation du MRTA

L'objectif du problème de MRTA est d'assurer la coordination des robots du système en répondant à la question suivante :

☐ Quel robot doit exécuter quelle tâche ?

Cette question est indépendante de tout contexte et fut appliquée à de nombreux problèmes, dont deux particlièrment recurents. Il s'agit de l'exploration et de la collecte, illustrées respectivement par les travaux de [27] et [60] en Figures 2.7 et 2.8.

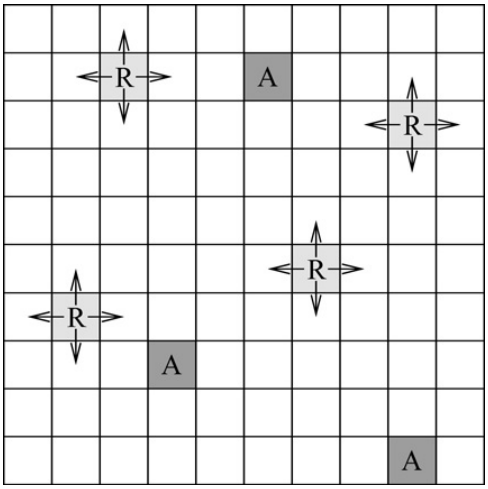

FIGURE 2.7 – Les robots (représentés par des R) doivent explorer les cases notées A. Ces dernières apparaissent périodiquement [27].

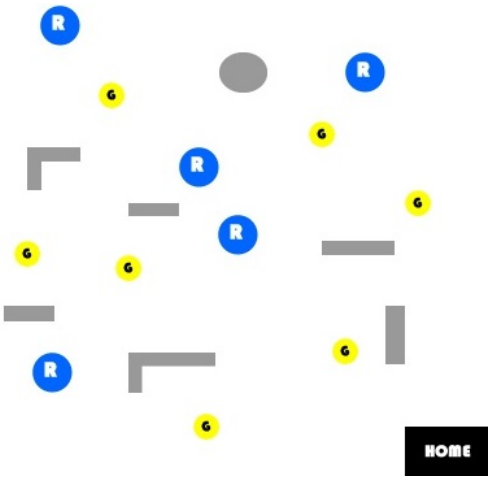

FIGURE 2.8 – Les robots (cercles bleus) doivent se répartir les tâches afin de collecter l'or (cercles jaunes) et le rapporter à la base (carré noir). [60]

Formalisation du problème MRTA

Cette diversité de contextes, situations et problématiques a poussé Gerkey et Mataric [6] à proposer une taxonomie du problème de MRTA afin de le formaliser. Cette dernière offre une classification rapide et indépendante du contexte selon les trois axes suivants :

1. Robot mono-tâche (ST) ou robot multi-tâche (MT) : les robots d'un système ST ne peuvent exécuter chacun qu'une tâche à la fois alors que ceux d'un système MT peuvent mener plusieurs tâches en parallèle.
2. Tâche mono-robot(SR) ou tâche multi-robot (MR) : dans un système SR, les tâches de type ST nécessitent seulement un robot pour être complétées alors que dans un système MR, certaines tâches nécessitent plusieurs robots.
3. Affectation instantanée (IA) ou affectation prolongée (TA) : les affectations IA sont effectuées sur la base d'informations ne permettant pas de planifier les futures allocations. Les allocations TA profitent de davantage d'informations comme la connaissance de l'ordre d'arrivée des tâches, leur nombre, etc.

Cependant, la pertinence d'une solution d'allocation de tâches dépend aussi fortement de l'architecture du MRS.

Architecture du système multi-robot

Cette architecture se caractérise selon deux axes majeurs :

1. Centralisée ou décentralisée : Dans un système centralisé, un agent central est responsable de la gestion de l'ensemble des ressources du système. Cette centralisation de l'information permet à l'agent central d'obtenir une vision globale de la situation, et donc, de trouver une solution optimale au problème. Cependant cette centralisation implique que tous les robots du systèmes communiquent avec l'agent central entraînant un fort coût de communications et la création d'un point de défaillance unique. À l'opposé, dans un système décentralisé, chaque robot prend lui même ses décisions basées sur une compréhension locale de la situation.
2. Homogène ou hétérogène : Dans un système homogène, les robots possèdent des caractéristiques identiques et sont donc capables d'exécuter les mêmes tâches. Lorsqu'un système devient hétérogène, certains de ses robots possèdent des spécificités allant de la présence (ou de l'absence) d'un capteur au changement de paradigme du robot

Dans ces travaux, nous nous intéressons aux architectures décentralisées (homogènes ou hétérogènes) pour lesquelles les approches basées sur le marché se révèlent efficaces.

Approches basées sur le marché

Les approches basées sur le marché s'inspirent directement des modèles économiques humains de type économie de marché, tels que les salles d'enchères et les places boursières. Dans ces systèmes, des individus motivés par leurs intérêts personnels échangent des ressources et services afin de maximiser leur profit. Ces échanges conduisent à la création d'un système de production global efficace.

Dans une approche basée sur le marché, chaque robot est motivé par son intérêt personnel et évolue dans une économie fictive où les tâches et ressources nécessaires à leur complétion deviennent des produits de consommation quantifiables pouvant être échangés. D'après la définition donnée par Dias et et al. [5], une approche basée sur le marché possède les pré-requis suivants :

1. Le MRS doit atteindre un objectif pouvant être décomposé en sous-objectifs réalisables par un ou plusieurs robots du système.
2. Le MRS possède un nombre limité de ressources pour compléter son objectif.
3. Il est possible de quantifier la qualité d'une solution à l'aide d'une fonction objectif globale.
4. Une fonction d'utilité spécifie pour chaque robot du système la valeur de ses ressources et leur contribution à l'objectif.
5. Les ressources et sous-objectifs peuvent-être échangés à l'aide d'un mécanisme de redistribution tel qu'une vente aux enchères.

Les enchères

Les enchères constituent le mécanisme d'attribution le plus répandu pour les approches basées sur le marché. Lors d'une vente aux enchères, un ensemble d'objets est mis en vente par l'intermédiaire d'un commissaire priseur. Chaque participant souhaitant acquérir un objet formule une offre (un prix). Le commissaire priseur attribue alors l'objet au participant ayant soumis la meilleure offre comme décrit en Figure 2.9. Cependant, un éventuel prix de réserve peut être défini. Dans ce cas, l'objet est attribué uniquement si l'offre excède le prix de réserve.

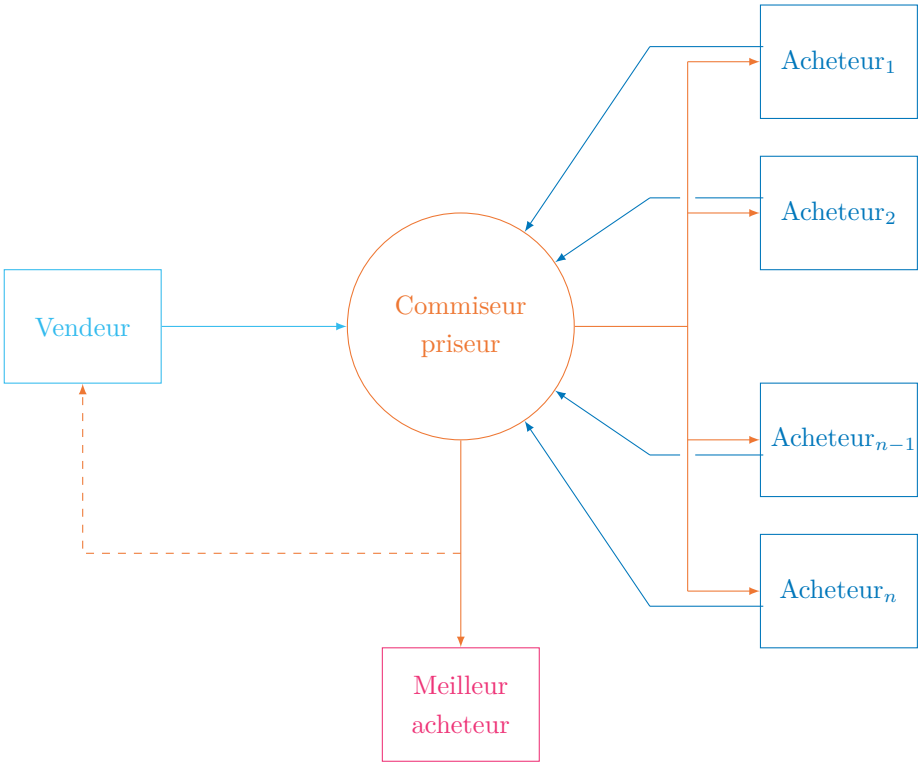

FIGURE 2.9 – Diagramme : le vendeur (cyan) transmet son offre au commissaire priseur (orange) qui la communique aux participants (bleu). Ces derniers formulent chacun une offre d'achat. Le commissaire priseur attribue alors l'objet de la vente au meilleur offrant (magenta) si la valeur de l'offre dépasse le prix de réserve

Il existe plusieurs types d'enchères :

1. Séquentielles : les objets sont mis en vente un par un. Cette approche rend la désignation du vainqueur simple puisqu'il suffit de sélectionner le meilleur enchérisseur. Cependant, lorsque l'enchérisseur souhaite acquérir un lot d'objets, il est alors obligé de spéculer sur les futures ventes. De plus, la séquence de vente influe sur la répartition. Pour plus de détails, se référer à [57].

2. Parallèles : l'ensemble des objets est mis en vente en parallèle. Cela ne résout pas le problème des lots puisque bien qu'étant en mesure d'enchérir simultanément sur plus objets, aucune garantie de recevoir l'ensemble des objets n'existe comme illustré en Figure 2.10.

3. Combinatoires : plusieurs objets sont mis en vente et chaque participant peut enchérir sur une combinaison d'objets. Cela permet de valoriser des synergies entre objets (particulièrement entre des tâches), mais se montre rapidement impraticable

à cause du nombre exponentiel de combinaisons à considérer [40].

FIGURE 2.10 – Le robot doit chercher quelle(s) tâche(s) nécessite(nt) le moins de déplacements par tâche. Dans le cas d'enchères séquentielles, le robot enchérira pour T_1 visant un ratio de 3 cases par tâche. En combinatoire, il enchérira pour T_2 et T_3 visant 2.5 cases par tâche. Par contre, que faire dans le cas d'enchères parallèles, viser T_2 et T_3 en prenant le risque de ne terminer qu'avec T_3 et son ratio de 5 cases par tâche ?

D'autres spécificités existent permettant d'affiner l'attribution. Ci dessous figure les variantes les plus utilisées dans l'univers de la robotique :

— Un tour vs multi-tours : lors d'enchères à un tour, les participants ne soumettent leurs offres qu'une fois. Il est impossible de surenchérir. *A contrario*, plusieurs tours d'enchère peuvent être autorisés, mais cela nécessite davantage de communications et allonge considérablement le processus décisionnel

— Ouverte ou cachée : dans le cas d'enchère cachée, seul le commissaire priseur connaît les valeurs des enchères proposées. Dans le cadre de MRS justes et coopératifs, les enchères ouvertes sont une source potentielle d'informations pour les robots, mais nécessite que les enchères soient diffusées à tous les participants.

— Premier ou second prix : dans le cadre d'enchères de type second prix, le gagnant ne paye que le second prix et non le sien (le premier prix). Cela encourage les fortes mises, mais ne possède aucun impact pour les systèmes justes comme attesté par [57].

Il a été démontré par Koenig [20] que les ventes aux enchères de type séquentielle constituent une solution adaptée à la coordination d'un MRS coopératif.

Les enchères doubles

Les ventes aux enchères étudiées précédemment s'avèrent efficaces pour sélectionner les acheteurs ou les vendeurs, mais elles ne permettent pas de mettre simultanément en concurrence les vendeurs et les acheteurs. Pour cela, il est nécessaire de recourir aux enchères doubles. En robotique, la mise en concurence directe des tâches est rarement recherchée. En effet si certaines tâches sont plus prioritaires que d'autres, elles doivent

néanmoins toutes être terminées. Il n'existe donc pas de raison d'écarter une offre de vente. Par conséquent, les enchères doubles sont rarement utilisées en robotique. Cependant, il s'agit d'un procédé utilisé en dehors de l'économie comme pour l'allocation de ressources en *grid* [16, 47], en *cloud computing* [39] ou le déchargement de données mobiles [12, 34]

Les enchères doubles, utilisées par les places boursières, mettent en scène plusieurs acheteurs et vendeurs au sein d'un unique marché de l'offre et de la demande (pour plus de détails, se référer à [29]). Contrairement aux enchères parallèles, il s'agit d'une vente globale et non de plusieurs ventes en parallèle. Le déroulement des enchères doubles est le suivant : chaque acheteur intéressé formule une offre d'achat et chaque vendeur propose un prix de vente. Puis le marché (le commissaire priseur) fixe un prix p devant garantir l'équilibre du marché. Tous les acheteurs qui ont proposé au moins p achètent et les vendeurs qui ont demandé moins de p vendent. Le prix p peut être fixé de façon à favoriser les acheteurs ou les vendeurs comme décrit en Figure 2.11.

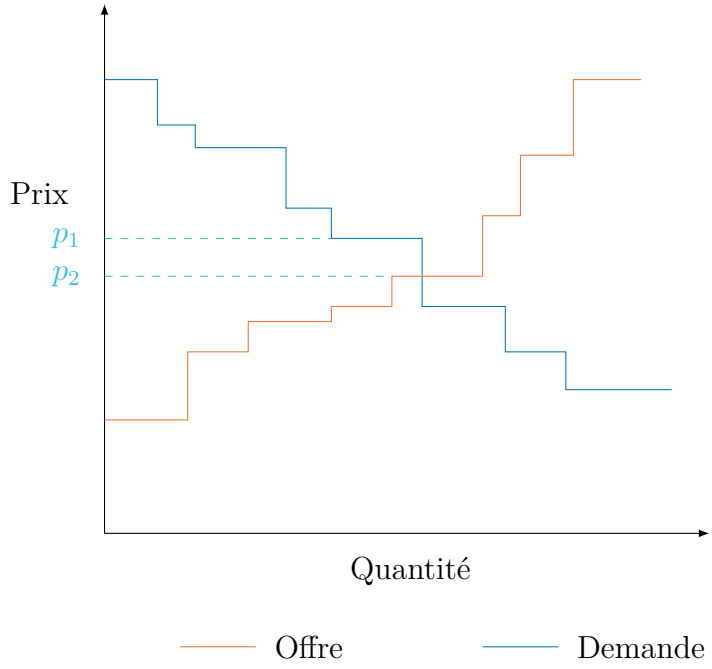

FIGURE 2.11 – Dans cet exemple représentant l'offre (en orange) et la demande (en bleu) selon des offres comprenant un prix pour une quantité souhaitée, l'équilibre du prix se situe entre p_1 et p_2. Il revient au marché de fixer un prix compris entre p_1, le prix de vente optimal et p_2, le prix d'achat optimal.

Le problème du MRTA a été largement étudié et de nombreuses solutions ont été proposées pour répartir le travail au sein de MRS décentralisées. Cependant, si de nombreuses solutions satisfaisantes existent, les tâches distribuées sont toujours considérées sous l'angle réduisant, bien souvent, le problème à une occurrence du problème du voyageur de commerce. Il existe parfois des spécificités, telles que des pertes de communication, mais une tâche est rarement envisagée sous son aspect calculatoire. Par conséquent, peu de travaux s'intéressent à la parallélisation/distribution d'un calcul haute performance au sein d'un MRS, notre objectif.

2.2.2 Calcul haute performance et MRS

Si le problème du MRTA délaisse la problématique calculatoire, cette dernière est pourtant bien présente. En effet, l'augmentation des architectures de capteurs et la complexification toujours croissante des algorithmes de traitement nécessaires à la prise de décision accroissent considérablement la charge calculatoire des robots, au point de nécessiter des ressources de traitement spécialisées (GPU, FPGA, etc.). Le domaine de la robotique propose deux approches pour répondre à ce besoin : le cloud robotique et le cluster robotique.

Cloud robotique

Suite au développement rapide des infrastructures cloud et du big data, l'intégration de ces technologies a permis aux systèmes MRS d'améliorer considérablement leurs performances et de gagner en complexité [17, 52]. Le recours au cloud affranchit les robots d'une partie des contraintes de l'univers embarqué en les transformant en capteurs/actionneurs mobiles. Le cloud robotique peut être défini de la façon suivante : un système robotique qui s'appuie sur des données ou du code obtenus depuis un réseau pour conduire ses tâches et dont toutes les données capteurs, tous les calculs ou toutes les mémoires ne sont pas intégrés directement dans le système. Le cloud robotique apporte les avantages suivants :

1. Calculs déportés : l'exécution des calculs sur des serveurs distants permet d'une part, d'alléger les robots de leurs tâches de calcul et, d'autre part, d'accélérer ces derniers en exploitant, entre autre, la parallélisation des calculs. Cela a notamment été employé pour des tâches de cartographie et localisation simultanées (SLAM) [36], de contrôle de formation [51] et d'analyse d'images [38]. Cependant, le recours au cloud computing entraîne une forte utilisation de la bande passante pour transférer les données nécessaires aux calculs.

2. Accès au Big Data : l'utilisation du cloud permet aux robots d'accéder à de larges bases de données qu'ils ne pourraient pas stocker localement compte tenu des limitations imposées par l'informatique embarquée. Son recours pour les tâches de saisie d'objet permet à la fois d'identifier l'objets à saisir [7], mais aussi d'améliorer la stabilité de la prise [18] comme illustré en Figure 2.12.
3. Apprentissage coopératif : l'utilisation du cloud offre la possibilité d'apprendre collectivement en partageant les expériences [2].

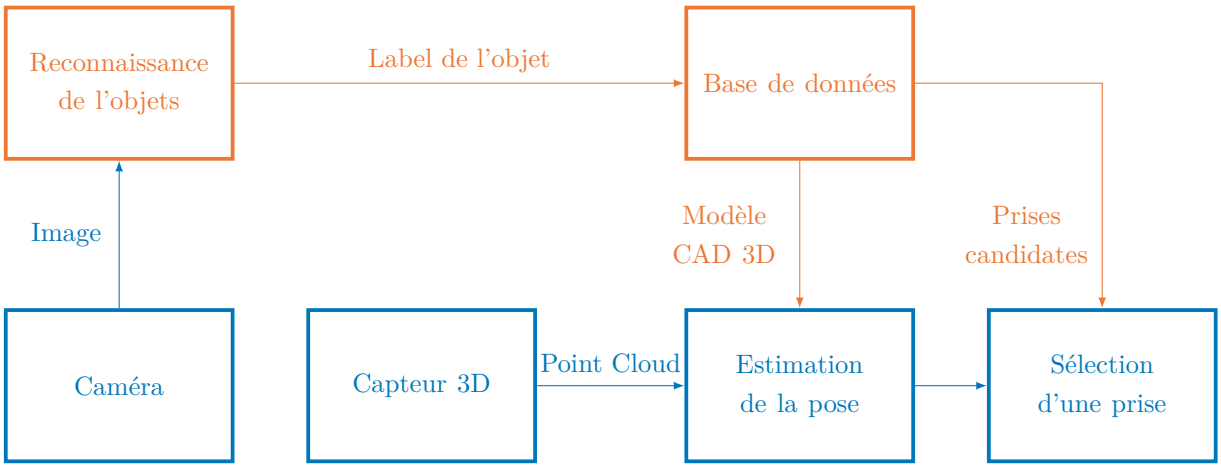

FIGURE 2.12 – Exemple d'utilisation du cloud (orange) pour assister la prise d'un objets par un robot (bleu). L'utilisation du cloud permet de détecter l'objet avec précision permettant de déterminer, à l'aide d'une base de données, les meilleures prises [18].

Si le recours au *cloud robotic* permet de s'affranchir des limites intrinsèques de l'informatique embarquée en déportant la charge calculatoire et en augmentant les capacités de stockage, il n'est pas dépourvu de limites.

1. La nécessité d'avoir accès à une bande passante stable et à haute capacité.
2. La présence d'un point de défaillance unique (l'accès au serveur). En cas de panne ou de rupture des communications, le MRS perd ses capacités calculatoires et ses données
3. La génération d'une latence dans les traitements rendant l'utilisation du cloud incompatible avec l'exécution de tâches où le temps de réponse s'avère critique (Ex : véhicules autonomes).

Cluster robotique

Constatant la nécessité pour certains MRS de posséder localement de la puissance de calcul, Marjovi et al. ont proposé le concept de cluster robotique [26] consistant, similairement aux clusters informatiques, à partager les ressources de calcul au sein d'un groupe de robots. Ils en donnent la définition suivante : un cluster robotique est un groupe composé de robots individuels capables de partager leurs ressources de calcul, au sein du groupe, afin de résoudre rapidement des problèmes à forte complexité calculatoire. L'objectif est de mettre en commun les ressources non utilisées par les membres du groupe comme décrit par la Figure 2.13. L'accélération d'une tâche par le cluster s'appuie sur sa parallélisation afin de la répartir entre les membres disponibles du cluster. Les tâches éligibles à une utilisation d'un cluster robotique sont celles qui nécessitent de forte ressources calculatoires, mais possèdent une faible complexité de mise en œuvre, telles que le traitement massif de données ou la cartographie d'une large zone.

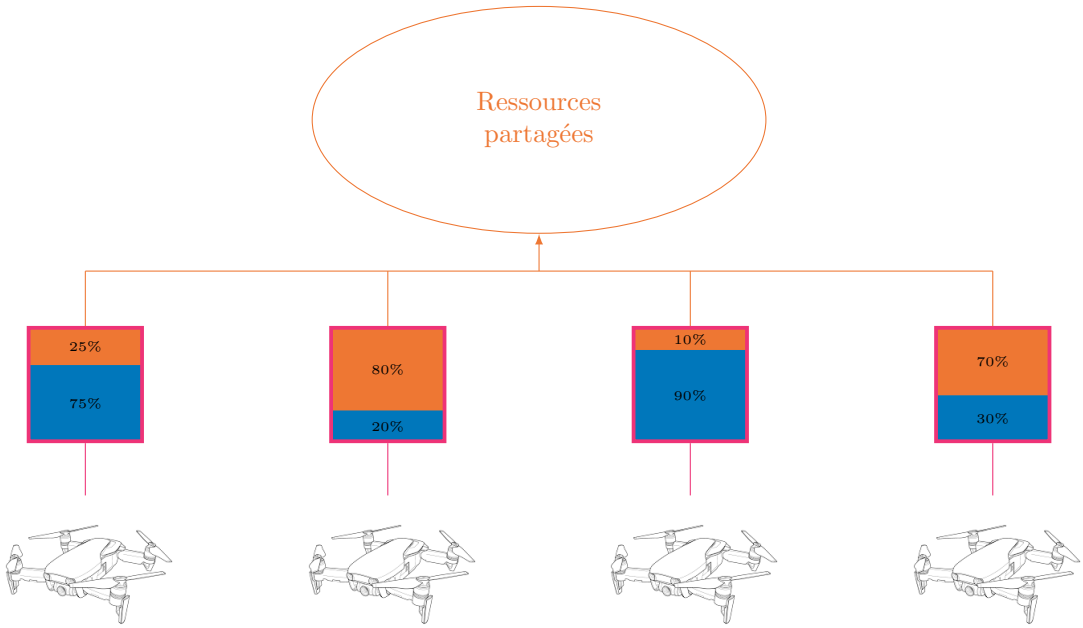

FIGURE 2.13 – Dans un cluster robotique chaque robot est défini par ses ressources de calcul (en magenta) divisées en deux parties. La première partie (en bleu) est constituée des ressources utilisées actuellement par le robot afin d'effectuer ses tâches courantes. La seconde partie (en orange) représente les ressources libres du robot. Ces dernières sont mises à disposition du cluster afin d'être partagées [26].

L'utilisation de cette méthode offre plusieurs avantages :

1. Meilleure utilisation des ressources de calcul : la plupart du temps les robots n'effectuent que des tâches simples n'exigeant pas beaucoup de ressources de calcul. Seuls quelques robots doivent compléter des tâches HPC. Par conséquent, les ressources calculatoires de nombreux robots sont sous-exploitées. Un cluster robotique utilise ces ressources afin d'aider les robots dans le besoin, augmentant alors les performances du système et optimisant l'utilisation de ses ressources.
2. Réduction des coûts : le déploiement d'un cluster robotique s'avère nettement moins coûteux que le déploiement d'un robot concentrant une puissance de calcul équivalente.
3. Évolutivité : contrairement à l'approche par robot unique où la puissance de calcul maximale est limitée, le cluster robotique peut être étendu en incorporant d'autres robots afin d'augmenter les capacités du cluster.
4. Fiabilité : aucun membre du cluster n'est crucial pour son fonctionnement. Par conséquent, cette approche se révèle moins sensible aux pannes.

Si cette méthode offre de nombreux avantages, son application nécessite certaines conditions qui limitent grandement les tâches pouvant tirer parti d'un cluster robotique. En effet, pour qu'une application soit éligible, elle doit respecter deux conditions :

1. posséder un fort degré de parallélisme : toutes les tâches ne peuvent tirer profit de la parallélisation. Afin de pouvoir être exécutée efficacement par un cluster, une tâche doit pouvoir être divisée en sous tâches et posséder le moins possible de parties séquentielles comme l'atteste la loi de Amdahl.
2. nécessiter le transfert de peu de données : le transfert massif de données génère un fort overhead et sature la bande passante du MRS nuisant à l'exécution des autres tâches. Il est donc important de trouver un équilibre entre accélération et transfert de données.

De plus les tâches nécessitant l'utilisation du cluster doivent représenter une part limitée des tâches effectuées par les robots. En effet, une utilisation massive du cluster le sature et rend la parallélisation inefficace.

Bien que présentant de nombreux avantages, le principe de cluster robotique a été peu utilisé. En effet, il n'existe, à notre connaissance, que deux autres études déployant

un cluster. La première étant celle de Gouveia et al. en 2015 où les calculs nécessaires au SLAM sont parallélisés sur un cluster de robots afin d'accélérer le traitement [8]. La seconde proposée par Camargo-Forero et al en 2018 [3], introduit la notion de calcul robotique haute performance (HPCR) rendue possible par la formation de clusters, mais demeure très théorique. Ces travaux attestent du fort potentiel calculatoire des clusters, mais aussi de leurs limites notamment lorsque le nombre de données à transférer est important.

Résumé

Apprentissage par renforcement

L'algorithme de *Q-learning* est une méthode de résolution *model-free*, *off-policy* et *"boostrapant"* qui apprend à partir d'expériences en estimant des paires (état, action) appelées *Q-values*. Cet algorithme fut amélioré avec l'introduction du *double Q-learning* qui résout un problème de surestimation.

Afin de pouvoir appliquer le *Q-learning* (et le *double Q-learning*) à des problèmes possédant un espace d'état à haute dimension, il est possible de recourir à un réseau de neurones pour estimer les *Q-values*. Cependant, cela entraine l'instabilité de l'algorithme à cause de deux facteurs :

1. La cible TD devient mouvante : cela peut être corrigé en déployant une copie gelée du réseau pour calculer la cible.
2. Il existe une forte corrélation entre deux expériences consécutives : l'utilisation d'un mécanisme de stockage d'expériences permet de briser cette corrélation en entrainant le réseau sur un sous-échantillon de cette mémoire. La sélection des échantillons s'effectue aléatoirement ou par hiérarchisation.

Plusieurs architectures ont été proposées avec notamment la 3DQN qui sépare les estimations de la *value function* et de l'*advantage function* offrant de meilleures performances et accélérant la convergence.

Cependant, une difficulté majeure persiste : le traitement des problèmes aux espaces d'actions à forte dimension. En effet, le nombre d'actions à considérer augmente expo-

nentiellement avec l'accroissement du nombre de dimensions. Cet obstacle est loin d'être surmonté, mais deux approches intéressantes existent :

1. L'approche multi-agents : cette solution décompose l'agent apprenant en plusieurs sous-agents. Chacun de ces agents traitent une dimension du problème et leur coordination est assurée par une récompense d'équipe. Cependant cette solution devrait faire face à un problème de convergence pour un grand nombre de dimensions.
2. L'approche par branches : cette architecture de réseau attribue une branche par dimension et agrège leurs sorties pour former l'action principale. Une dernière branche, commune à l'ensemble des dimensions, estime la *state-value function* et facilite la convergence.

Distribution et parallélisation de tâches

La distribution des tâches au sein d'un système multi-robots conditionne la coordination de ce dernier. Par conséquent, il s'agit d'une problématique cruciale qui reçut une forte attention et de nombreuses solutions furent proposées. Cependant, la pertinence d'une solution dépend fortement du contexte, et notamment, de l'architecture du système multi-robots.

Dans le cas d'un système décentralisé, les approches basées sur le marché se révèlent pertinentes. Elles consistent à faire évoluer les robots dans une économie fictive où les tâches et ressources nécessaires à leur exécution s'échangent. Chaque robot cherche alors à maximiser son profit conduisant à l'amélioration des performances du système. Couramment, le mécanisme d'échanges s'appuie sur un système d'enchères pour effectuer les attributions. De nombreux types d'enchères existent et ils répondent à des besoins particuliers. Si l'objectif recherché est la mise en concurrence simultanée des acheteurs et vendeurs, il est alors nécessaire de recourir à des enchères doubles qui s'apparentent à des offres boursières. Cependant, quelle que soit l'approche de résolution utilisée, les occurrences actuelles de MRTA ne traitent pas de problématiques relatives à la parallélisation (et la distribution) des tâches fortement calculatoires.

Ces dernières sont abordées indépendamment du MRTA avec les concepts de cloud et de cluster robotique :

1. Cloud robotique : cette solution consiste à déporter les calculs, les données et parfois même la prise de décision à un système extérieur. Ce dernier possède des capacités

nettement supérieures à celles de la MRS et libère les robots d'une grande partie de contraintes liées à l'univers embarqué. Cependant, cette solution nécessite l'accès à une bande passante stable et à haute capacité. De plus, elle entraîne la génération d'une latence dans les traitements. Par conséquent, bien qu'efficace, cette approche est inutilisable dans de nombreuses situations.

2. Cluster robotique : les limites du cloud robotique ont conduit au développement d'une approche de traitement locale au MRS. Elle consiste à mettre en commun les ressources non-utilisées du système afin d'accélérer les robots dans le besoin. Cette solution ne souffre pas des limitations du cloud robotique, mais elle n'est applicable qu'aux tâches fortement parallélisables et nécessitant peu d'échanges de données.

Introduction aux travaux

Le contexte de ces travaux est celui d'un système multi-robots homogène évoluant dans un environnement dynamique et incertain. Nous avons donc opté pour une architecture décentralisée où l'absence de point de défaillance unique s'avère vitale. De plus, nous considérons des situations où le système ne peut compter que sur lui même, aucune aide extérieure n'étant possible. Par conséquent, le recours au cloud robotique s'avère impossible.

Dans ces conditions, nous avons étudié les conséquences de l'apparition de tâches fortement calculatoires au sein d'un système multi-robots, un problème critique, mais encore peu exploré. Afin de maximiser les ressources disponibles, notre système multi-robot forme un cluster robotique mutualisant ainsi ses capacités calculatoires. La gestion d'un tel cluster nécessite des approches adaptatives et dynamiques par conséquent, nous avons analysé l'emploi de méthodes utilisant l'apprentissage par renforcement pour répondre à ce besoin. Nous avons traité les deux axes de travaux suivants :

1. Dans le chapitre 3, nous explorons l'emploi d'une solution utilisant l'apprentissage par renforcement comme substitut aux approches basées sur le marché dans le cadre d'une variante du MRTA.
2. Dans le chapitre 4, nous présentons plusieurs solutions basées sur le *deep Q-learning* pour administrer dynamiquement un cluster robotique dans le cadre d'un contexte de recherche et sauvetage.

Chapitre 3

Allocation de tâches calculatoires au sein d'un cluster robotique

Sommaire

3.1 Introduction

La complexité toujours croissante des architectures de capteurs et des algorithmes de traitement nécessaires à la prise de décision entraîne une forte augmentation de la charge calculatoire devant être supportée par les robots. Si le cloud robotique permet aux robots de se libérer de cette charge, il est inapplicable dans de nombreuses situations critiques où la latence est importante (ex : conduite autonome) et/ou les communications demeurent incertaines (ex : zones de sinistres). Par conséquent, dans ces circonstances, le système multi-robots ne peut compter que sur ses ressources et doit prendre en considération le coût calculatoire d'une tâche dans son processus d'allocation afin de répartir la charge.

Le système multi-robots peut alors former un cluster robotique afin de mutualiser les ressources disponibles. Ainsi, en cas de surcharge, un robot du système pourra tenter de transférer ses tâches excédentaires à d'autres membres. Cependant, si ce transfert maximise l'utilisation des ressources du système, il monopolise des ressources et peut donc s'effectuer au détriment d'une future tâche plus prioritaire. En effet, contrairement au cloud robotique, les ressources d'un cluster robotique s'avèrent intrinsèquement limitées. De plus, le recours au transfert génère de l'*overhead*. Il est donc nécessaire de trouver un juste équilibre entre maximisation du transfert, stockage des ressources pour des tâches prioritaires et minimisation de l'*overhead*.

Devant ce constat, nous définissons un nouveau problème le *Multi-Robot processing Task Allocation* (MRpTA) qui constitue une variante du célèbre MRTA. Contrairement à ce dernier, les contraintes majeures sont les ressources calculatoires et non la position des robots. Dans ce contexte, nous proposons une première approche de résolution où les robots peuvent échanger des tâches calculatoires à l'aide d'un mécanisme de transfert. Compte tenu de la nature dynamique du problème, une approche utilisant l'apprentissage par renforcement semble être un candidat naturel.

Ainsi, dans ce chapitre, nous explorons la viabilité d'une méthode d'apprentissage par renforcement de type *deep Q-network* pour résoudre un problème de MRpTA en répondant aux questions suivantes :

1. Les robots d'un système décentralisé peuvent-ils apprendre par eux-mêmes à allouer efficacement les tâches du système ?
2. Quelles sont les performances d'une approche utilisant l'apprentissage par renforcement comparées à celles obtenues par une approche basée sur le marché ?

Nous allons maintenant définir les concepts clés de notre problème du MRpTA.

3.2 Modélisation du problème

Les approches étudiées partagent une même définition de l'environnement. Elles diffèrent uniquement par leur processus de décision, notamment en ce qui concerne l'équilibrage de la charge au sein du système.

3.2.1 Définition de l'environnement

Définition du système multi-robots

Le MRS poursuit un double objectif : la complétion d'un maximum de tâches qui lui sont assignées tout en favorisant les tâches prioritaires. Pour cela, le système doit répartir la charge à l'aide du mécanisme de transfert. Cependant, les contraintes imposées par l'environnement rendent impossible le respect des deux objectifs. Le système doit faire un choix.

Ces choix s'effectuent dans un contexte de MRS décentralisé, par conséquent, chaque robot effectue ses propres choix basés uniquement sur sa connaissance locale du système. Les robots communiquent uniquement lors d'un transfert, aucun autre échange d'information n'a lieu. Afin de s'approcher davantage de conditions réelles, un système de zone est défini. Ce dernier limite les transferts aux robots d'une même zone. Si les robots peuvent changer de zone en cours de mission, cela ne dépend pas de leur volonté et ne figure pas dans les actions qu'ils peuvent entreprendre. Ces dernières, au nombre de trois, affectent uniquement des tâches et sont les suivantes :

1. Exécute : le robot exécute la tâche. Pour cela, il doit posséder suffisamment de ressources calculatoires disponibles.
2. Reporte : le robot reporte la tâche qui, si elle n'a pas atteint sa laxité, sera présente dans la file de tâche du robot à la prochaine itération, sinon elle est considérée comme échouée (et disparaîtra de la file de tâche).
3. Transfert : Tente de transférer la tâche à un autre robot de sa zone. En cas d'échec, la tache est automatiquement reportée. Le transfert d'une tâche implique la génération d'un *overhead*.

Les pré-requis des différentes actions dépendent des caractéristiques de la tâche sur laquelle le robot souhaite agir.

Définition d'une tâche

Les tâches sont indépendantes (les tâches possédant des dépendances sont regroupées en une seule tâche) et possèdent une priorité fixe, mais aucune tâche n'est impérative. Similairement à des travaux précédents [9], nous considérons que les caractéristiques d'une tâche sont connues lors de son arrivée dans la file du robot. Les caractéristiques principales d'une tâche sont les suivantes :

TABLE 3.1 – Caractéristiques principales d'une tâche

Caractéristique	**Définition**	**Valeur**
CPU	Quantité de ressources CPU nécessaire à l'exécution de la tâche	Entier $0 \rightarrow 100$
Mémoire	Quantité de mémoires nécessaire à l'exécution de la tâche	Entier $0 \rightarrow 100$
Temps d'exécution	Nombre d'itérations nécessaires à la complétion de la tâche	Entier $1 \rightarrow n$
Priorité	À sa création, chaque tâche reçoit une valeur de priorité fixe sélectionnée aléatoirement de façon uniforme	1 : haute 2 : moyenne 3 : faible
Laxité	Le nombre maximum d'itérations avant le commencement de la tâche	Entier $1 \rightarrow l$

Modélisation du temps

Afin de pouvoir simuler le comportement du système, il est nécessaire de discrétiser la temporalité, la diviser en itérations de durées équivalentes comme illustré en figure 3.1. De plus, dans un souci de performance, la granularité de la discrétisation ne doit pas être trop fine pour ne pas générer un nombre important de pas temporels inutiles (sans changement). Cela implique les conséquences suivantes :

1. Les robots reçoivent simultanément leurs nouvelles tâches et les traitent en parallèles.
2. Plusieurs tâches commencent ou terminent exactement au même moment comme décrit en figure 3.1.b.
3. Les transferts de tâche interviennent après les allocations locales, mais dans la même itération.

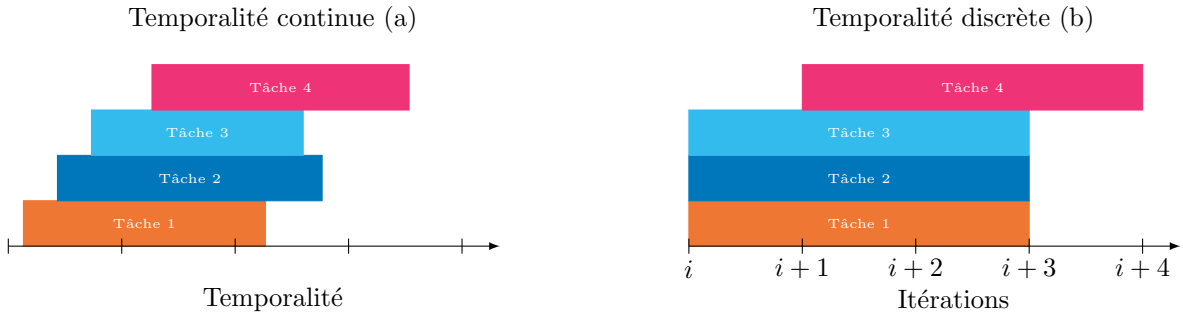

FIGURE 3.1 – Temporalité : la figure de gauche représente une temporalité continue et celle de droite une temporalité discrète. Contrairement à la temporalité continue, le cas discret comporte des tâches possédant un temps d'exécutions identiques (3 cycles) et les tâches n°1,2 et 3 commencent et terminent en même temps. Cette discrétisation implique une standardisation des temps d'exécution et de la simultanéité.

Les nouvelles tâches générées sont stockées dans une file et traitées séquentiellement, un tri par priorité décroissante puis par *deadline* la plus proche est effectué à chaque itération. Le processus décisionnel entrant alors en action dépend de la solution utilisée.

3.2.2 Solutions

Nous proposons deux types de solution pour ce problème du MRpTA. Le premier, basé sur le marché, est couramment utilisé pour résoudre des problèmes de MRTA.

Approches basées sur le marché

Nous avons considéré deux solutions pour cette approche, une avec préemption et une sans préemption. Elles partagent le même processus décisionnel, seul le mécanisme de transfert diffère.

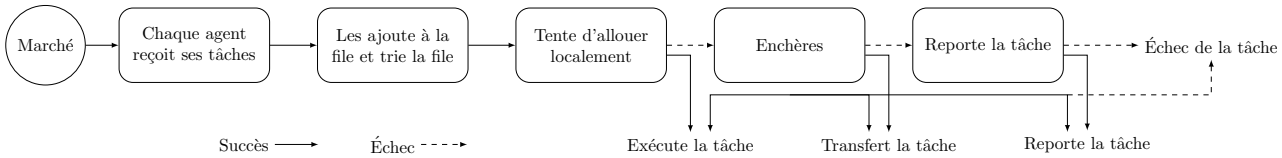

FIGURE 3.2 – Le processus décisionnel de la méthode par marché est fixe. Ainsi cette solution essaiera d'abord d'allouer localement puis en cas d'échec formulera une proposition de transfert et finalement, en cas de nouvel échec reportera, la tâche.

Lorsqu'un agent souhaite transférer une tâche, il devient vendeur et soumet alors une offre au commissaire-priseur de la zone qui la diffuse aux autres agents présents. Ces derniers formulent alors une offre d'achat selon le processus décrit par l'algorithme 1. La tâche est ensuite attribuée au meilleur enchérisseur. La valeur d'une offre d'achat dépend des ressources calculatoires disponibles de l'agent afin de favoriser les robots en "sous-charge" et équilibrer la charge au sein du système. De surcroît, dans l'approche préemptive, un agent peut arrêter l'exécution de certaines tâches afin d'accepter une tâche plus prioritaire. Les tâches ainsi arrêtées pourront être redémarrées, mais depuis le début et en respectant leurs laxités. De plus, dans cette approche préemptive, le vendeur participe aussi en temps qu'acheteur. Cela permet de favoriser un transfert de la tâche plutôt que son exécution locale avec préemption. Il en va de même pour la valorisation d'une offre qui privilégie l'absence de préemption et plus particulièrement l'absence de préemption sur des tâches de priorités moyennes. Ainsi, la valeur d'une offre n'entraînant pas d'éviction sera toujours supérieure à celle en entraînant.

Il convient de mentionner que l'utilisation des enchères ne contredit pas la nature totalement décentralisée de notre système. En effet, cette centralisation est temporaire et tout agent peut devenir commissaire-priseur [1]. Par conséquent, aucun point de défaillance unique n'est introduit.

1. Une approche simple mais efficace consiste à attribuer ce rôle de commissaire-priseur au robot souhaitant transférer une tâche, le temps de sa vente

Algorithm 1: Algorithme de valorisation d'une enchère
(En orange, les étapes spécifiques à l'approche préemptive)

Data: T, la tâche mise aux enchères
A, la liste des tâches en cours d'exécution sur le robot
A_i, la liste des tâches en cours d'exécution de priorité i
R_{cpu}, la charge de CPU libre du robot
R_{mem}, la quantité de mémoire libre du robot
Function :
cpu(i), retourne la quantité de CPU nécessaire à l'exécution de la tâche i
mem(i), retourne la quantité de mémoire nécessaire à l'exécution de la tâche i
Result: Bid, la valeur de l'offre
if $R_{cpu} \geqslant cpu(T)$ *and* $R_{mem} \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem}$;
else if $priority(T) = 1$ *and* $R_{cpu} + \sum_{i \in A \setminus A_1} cpu(i) \geqslant cpu(T)$ *and*
$R_{mem} + \sum_{i \in A \setminus A_1} mem(i) \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem} - \sum_{i \in A_1} (\text{cpu}(i) + \text{mem}(i))$
else if $priority(T) = 2$ *and* $R_{cpu} + \sum_{i \in A_3} cpu(i) \geqslant cpu(T)$ *and*
$R_{mem} + \sum_{i \in A_3} mem(i) \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem} - \sum_{i \in A \setminus A_3} (\text{cpu}(i) + \text{mem}(i))$
else
  $Bid = -\rho$;
end

Si l'utilisation du marché possède de nombreux avantages, il nécessite de pouvoir évaluer correctement le prix d'une offre. Or, la complexification toujours croissante des systèmes rend cette valorisation difficile. Devant ce constat, nous proposons une approche dynamique à base d'apprentissage par renforcement.

Approche utilisant un DQN

Contrairement à l'approche précédente, cette solution définit librement son processus de décision, sa seule contrainte étant la validité de l'action comme illustré en figure 3.3. En effet, le système ne peut choisir une action impossible comme exécuter une tâche sans atteindre les prérequis nécessaires. De plus, aucune préemption n'est autorisée pour deux

raisons. D'une part, nous souhaitons que le système apprenne à modérer ses allocations afin de conserver suffisamment de ressources pour des tâches plus prioritaires. D'autre part, cela nécessiterait de complexifier fortement l'environnement afin d'intégrer toutes les données nécessaires à la prise de décision et de rester proche d'un MDP.

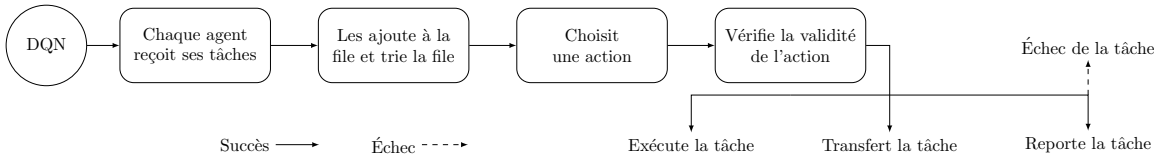

FIGURE 3.3 – Le processus décisionnel de l'approche par renforcement dépend de la politique apprise. Aucune restriction autre que la validité d'une action n'est imposée à cette approche

Cette prise de décision est assurée par un *deep Q-network* dont l'architecture est décrite en figure 3.4. Afin de respecter les contraintes imposées par l'univers embarqué, nous avons déployé une architecture de taille raisonnable (attestée par nos expérimentations cf. Tableau 3.2) pouvant fonctionner sur un robot sans monopoliser ses ressources ce qui le rendrait inapte à conduire sa mission. Comme décrit dans le chapitre précédent, nous utilisons les mécanismes de *target network* et *d'experience replay* afin d'assurer la stabilité de l'algorithme dont le paramétrage est donné en section 3.3.1. Nous avons déployé un mécanisme de contrôle des actions prises par le réseau afin d'empêcher la prise d'actions interdites (ce qui peut notamment arriver lors de la phase d'exploration). En cas de sélection d'une action impossible, cette dernière est annulée, mais une expérience est créée avec une valeur de récompense de -1 rendant toute action possible plus profitable. Mis à part ce processus de contrôle, le DQN est entièrement responsable de la prise d'actions de cette solution.

Si les effets des actions sont similaires pour les deux approches, le transfert diffère légèrement. En effet, contrairement à l'approche basée sur le marché, aucune vente aux enchères n'est réalisée. Par conséquent, comme aucune offre n'est émise par les agents répondant favorablement à la demande, l'agent transférant ne peut évaluer la pertinence des acceptations, et donc, choisir le meilleur candidat. En effet, si plusieurs agents répondent favorablement, le gagnant sera sélectionné aléatoirement.

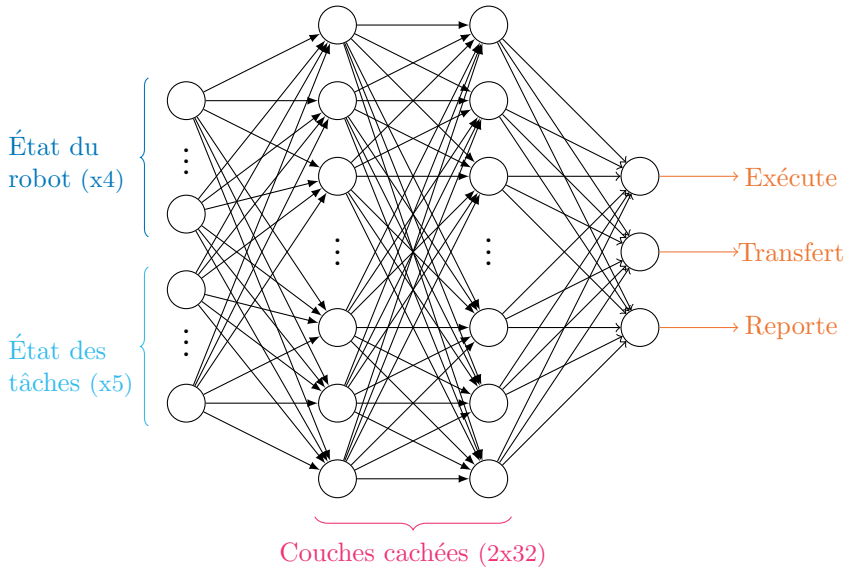

FIGURE 3.4 – L'architecture utilisée est un DQN composé d'une couche d'entrée de neuf neurones dont quatre traduisent l'état du robot et cinq l'état de la tâche sur laquelle portera l'action. Viennent ensuite deux couches cachées totalement connectées composées de 32 neurones. Puis, la couche finale contenant un neurone par action possible. Ces paramètres ont été obtenus par essais successifs afin de parvenir au meilleur ratio performance/taille possible.

Pour finir, le système de récompense employé poursuit les deux objectifs de la mission à savoir la complétion d'un maximum de tâches et le respect des priorités ainsi qu'un troisième objectif, la limitation de l'*overhead* généré par l'utilisation du transfert. Pour cela, nous récompensons différemment les actions selon qu'elles conduisent ou non à l'exécution d'une tâche.

$$r(s,a) = \begin{cases} V_i, & \text{si exécutée.} \\ \dfrac{j \times (3-i)}{L} + \beta\delta_{0\tau}, & \text{sinon.} \end{cases} \tag{3.1}$$

où :

— i, j, L représentent respectivement la priorité de la tâche, le nombre de report déjà effectuées et la laxité de la tâche.

— $V = [\alpha_1, \alpha_2, \alpha_3]$: un vecteur de constantes correspondant aux poids des priorités.

— τ : prend la valeur 0 si le transfert est un succès et 1 sinon.

— β : une constante servant à pénaliser le transfert.

3.3 Résultats

3.3.1 Conditions d'expérimentation

Notre problème met en jeu un système multi-robots totalement décentralisé capable d'exécuter de multiples tâches calculatoires et de former un cluster robotique pour transférer des tâches. Un tel système n'existe pas encore et nous avons donc dû concevoir un simulateur afin d'attester de la pertinence du problème du MRpTA et de valider nos propositions.

Présentation du simulateur

Notre simulateur est codé en python et le déploiement du DQN est assuré par le framework (open source) d'apprentissage automatique Tensorflow (avec Keras). Il est capable de :

1. Simuler le déploiement d'un système multi-robots décentralisé dans un contexte de MRpTA.
2. Produire plusieurs variations du problème en modulant des paramètres clés :
   (a) Caractéristiques de l'environnement (probabilités de changement de zone et système de génération des tâches).
   (b) Le jeu de tâches utilisé.
   (c) Configuration du système multi-robots (taille, capacité de calcul, homogénéité/hétérogénéité, etc.)
3. Offrir deux types de résolutions :
   (a) Classiques s'appuyant sur un système d'enchères pour répartir la charge
   (b) Basé sur l'apprentissage par renforcement au moyen d'une architecture DQN.

Bien que simulant notre problème, nous avons souhaité nous assurer de la viabilité du déploiement de notre solution "DQN" sur un robot mobile. Pour cela, nous avons testé les temps d'inférence et d'apprentissage sur une architecture matérielle dédiée à l'informatique embarqué. Les résultats sont présentés dans le tableau 3.2. Comme attendu, l'architecture minimaliste que nous avons développée peut parfaitement être utilisée dans le cadre d'un déploiement réel. Elle ne représentera qu'une partie infime de la charge calculatoire et l'apprentissage en ligne est tout à fait envisageable.

TABLE 3.2 – Temps d'inférence et d'apprentissage selon l'architecture

Configuration	Emulateur	Nvidia Jetson TX2
Composants	Intel Xeon Silver 4114 (x2) Nvidia GTX 1080 TI	ARM Cortex-A57 Nvidia Denver 2 GPU Pascal
Capacité	20 coeurs cadencés à 2.2 GHz 64 GB de DDR4 3584 coeurs CUDA 11 GB de GDDR5X	4 coeurs cadencé à 2 GHz+ 2 coeurs cadencé à 2Ghz 8 GB de LPDDR4 256 coeurs CUDA
Durée d'une inférence	2ms	4.5ms
Durée de l'apprentissage d'un batch de 24 expériences	210ms	520ms

Performances de référence

Afin de faciliter l'analyse des résultats, nous avons intégré aux tests deux solutions de comparaisons supplémentaires qui font office de borne inférieure et de borne supérieure.

1. Solution "borne inférieure" : une approche totalement décentralisée où les robots sont indépendants et ne communiquent pas entre eux. Chaque robot exécute ses tâches localement.
2. Solution "borne supérieure" : nous avons logiquement choisi une approche centralisée qui profite d'une vision globale du système. Par conséquent, les approches développées doivent tenter de s'approcher des résultats obtenus par cette méthode. Comme le système est réparti sur plusieurs zones, il existe un agent central par zone. À chaque itération, les tâches qui devraient normalement être attribuées aux agents d'une zone sont, à la place, assignées à l'agent central de cette zone. Ce dernier les alloue ensuite aux agents présents (sans surcoût). Les tâches non allouées sont conservées par l'agent central afin d'être redistribuées lors de l'itération suivante. Identiquement à l'approche préemptive par marché, l'agent central peut forcer l'exécution de tâches plus prioritaires.

L'ensemble des processus décisionnel est illustré dans le schéma 3.5. Pour l'ensemble de cette section, les termes "DQN", "Marché", "Marché-P, "Borne inf." et "Borne sup." feront respectivement référence aux solutions par DQN, marché sans préemption, marché avec préemption, borne inférieure et borne supérieure.

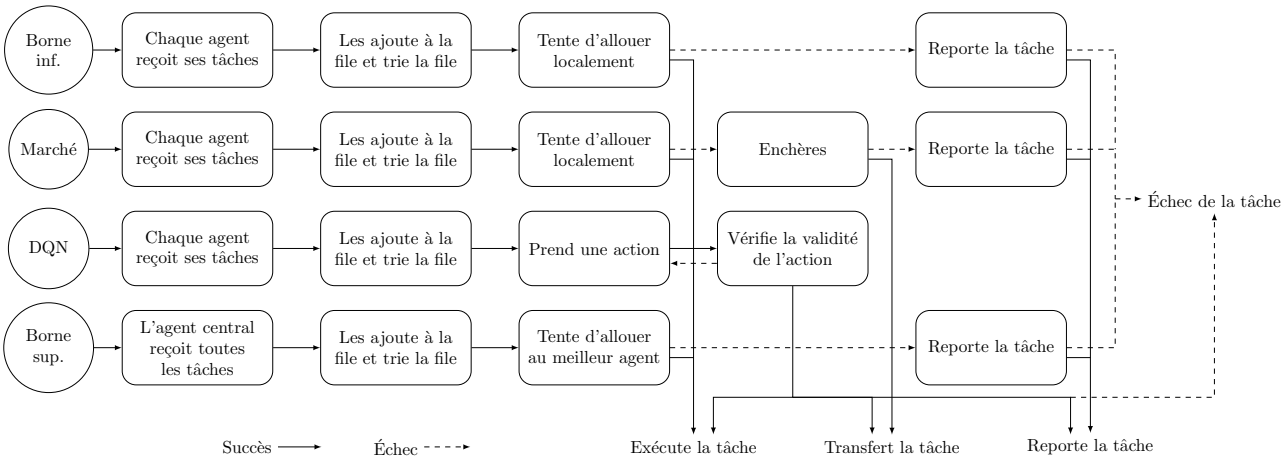

FIGURE 3.5 – Résumé de l'ensemble des processus décisionnels

Configuration

Le nombre important de configurations différentes, nous a conduit à restreindre notre étude à un ensemble de configurations où de nombreux paramètres sont fixés. Ces paramètres obtenus par essais successifs sont les suivants :

TABLE 3.3 – Paramètres du DQN et de l'environnement

Paramètre du DQN	**Valeur**	**Paramètre de l'environnement**	**Valeur**
γ	0.5	Nombre de robots	7
ε *decay*	0.995	Nombre de zones	3
α	10	Report maximum d'une tâche	5
c	40	Durée d'exécution d'une tâche	4
Taille de la mémoire	256	Nombre de priorités	3
Taille du sous-échantillon	24	Pondération du choix des priorités	(1, 1, 1)
$V = [\alpha_1, \alpha_2, \alpha_3]$	$[1, 0.3, 0.05]$		

De même, il existe de nombreuses manières de concevoir la génération de tâches. La plus pertinente semble être une génération de tâches apériodiques où des pics d'activités apparaissent afin de mettre en évidence des périodes de surcharge où des choix doivent être effectués. Pour modéliser cela, nous utilisons une suite périodique où le nombre de

tâches assignées au système multi-robots [2], U_i, pour une itération i est le suivant :

$$U_i = 12 - |10 - p| \text{ où } i \equiv p \; mod \; 20 \tag{3.2}$$

Concernant les zones de répartitions des robots, elles sont au nombre de trois (Z_1, Z_2 et Z_3). Toutes les dix itérations, tous les robots se voient attribuer une zone (qui peut être la même que précédemment) selon les probabilités suivantes : $P(Z_1) = \frac{1}{2}$, $P(Z_2) = \frac{1}{3}$ et $P(Z_3) = \frac{1}{6}$. Ce déséquilibre dans la répartition induit des capacités de transfert différentes selon les zones forçant les solutions à s'adapter.

Pour terminer, sauf mention contraire, le jeu de tâches utilisé est celui décrit dans le tableau 3.4. Il s'agit d'un jeu équilibré où, en moyenne, une tâche nécessite (en pourcentage) autant de CPU que de mémoire.

Table 3.4 – Jeu de tâches équilibré

Tâches								
ID	CPU (%)	RAM (%)	ID	CPU (%)	RAM (%)	ID	CPU (%)	RAM (%)
1	5	5	9	15	10	16	10	15
2	10	10	10	25	15	17	15	25
3	15	15	11	30	5	18	5	30
4	20	20	12	30	20	19	20	30
5	25	25	13	35	20	20	20	35
6	30	30	14	35	25	21	25	35
7	35	35	15	40	10	22	10	40
8	40	40						

Compte tenu du nombre important d'aléas lors de la génération de tâches, les résultats présentés sont une moyenne de ceux obtenus sur **1000 simulations** de **500 itérations** continues chacune. De plus, pour chaque simulation, les tâches générées et changement de zones sont identiques pour toutes les solutions. Concernant l'apprentissage de l'approche DQN, nous l'avons entrainée préalablement pendant 300 itérations.

2. Chaque tâche est ensuite assignée à un robot choisi de façon aléatoire et équiprobable.

3.3.2 Performances globales

Ces résultats ont pour objectif de répondre à des questions posées par l'utilisation de ces approches.

☐ La première question est : comment l'utilisation de ces solutions affecte la capacité du système à compléter les tâches qui lui sont assignées ?

Taux de complétion

La figure 3.6 présente le nombre de tâches complétées et échouées pour chaque solution. Nous constatons que les approches "Marché-P" et "Borne-sup" offrent un taux de complétion inférieur. Cela s'explique par la nécessité de recommencer les tâches à zéro lorsque la préemption est utilisée. À l'opposé, la solution "Marché" tire profit du mécanisme de transfert et réalise le meilleur taux de complétion global.

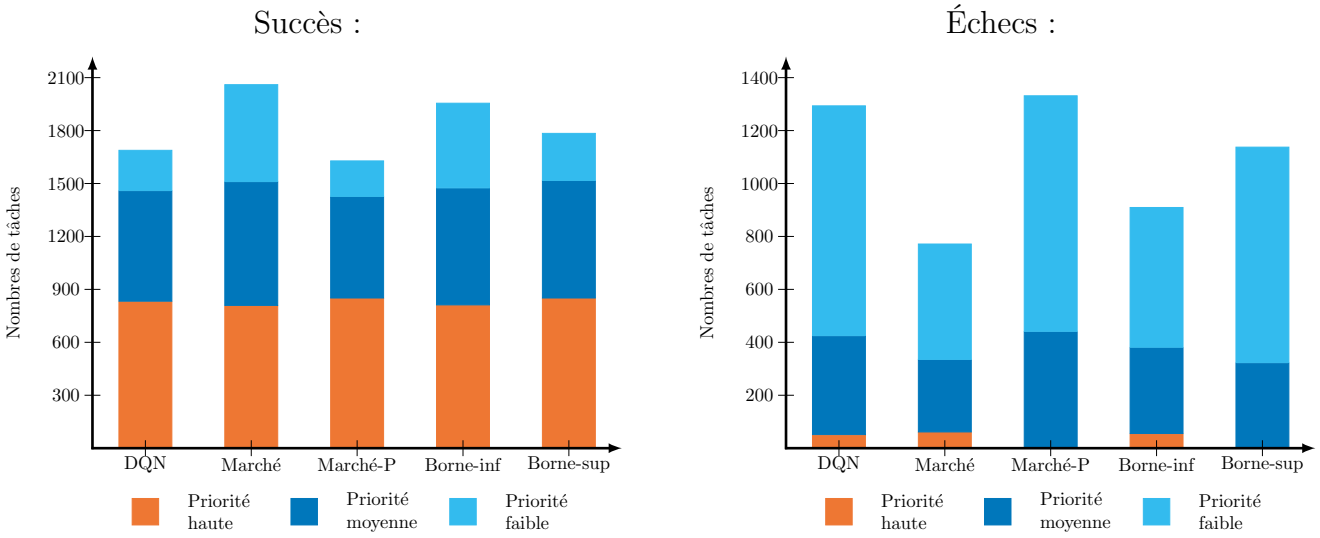

FIGURE 3.6 – Le graphique de gauche présente le nombre de tâches complétées selon leur niveau de priorité par chaque solution et celui de droite, le nombre de tâches échouées.

☐ Les approches utilisant la préemption complètent moins de tâches. Sont-elles capables de se montrer meilleures dans la gestion de la "criticité" des tâches ?

Gestion de la "criticité"

Comme attendu, les approches préemptives respectent parfaitement le système de priorités en assurant la totalité des tâches à haute priorité. L'approche par renforcement offre

de meilleurs résultats que les approches non-préemptives, mais ne parvient pas à garantir l'exécution des tâches de priorité forte. Néanmoins, cela démontre que cette solution se réfrène d'allouer toutes ses ressources afin d'en conserver pour des tâches plus prioritaires.

Nous constatons un compromis puisque le système ne peut à la fois maximiser son taux de complétion et respecter les priorités. La pertinence d'une solution dépend donc de l'importance apportée à ces deux facteurs.

☐ Dans cette situation, comment évaluer les performances des différentes solutions ?

Répartition de la charge

La capacité d'une solution à maximiser l'utilisation des ressources disponibles constitue un bon indicateur de performances. La figure 3.7 illustre la charge moyenne des robots de

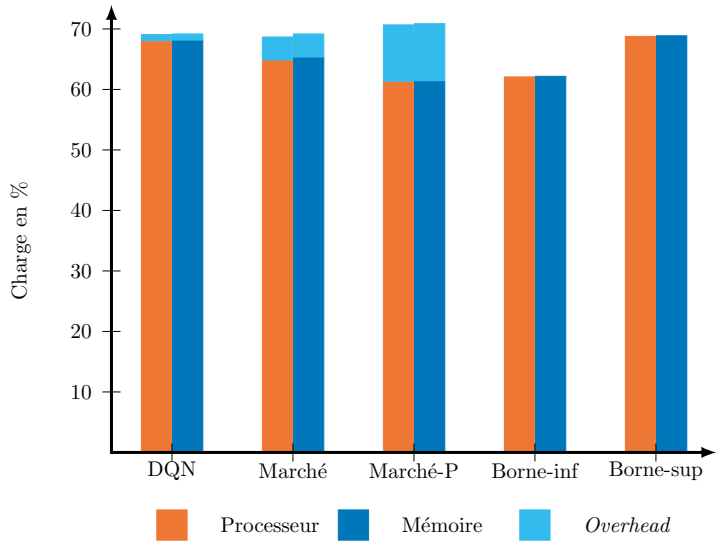

FIGURE 3.7 – Taux d'utilisation moyen des ressources calculatoires de chaque robot en fonction de la solution utilisée. La partie colorée en cyan représente l'overhead généré par l'utilisation du mécanisme de transfert.

chaque approche avec, en orange, la charge CPU, en bleu, le pourcentage de la mémoire utilisée et, en cyan, les ressources dépensées pour l'*overhead* généré par les transferts. Sans surprise, la borne supérieure, profitant de sa vision globale, présente la meilleure utilisation des ressources. À l'opposé, nous retrouvons la borne inférieure qui dépourvue de mécanisme de répartition de la charge fournit les pires résultats. Des trois approches

distribuées, la solution "DQN" propose le schéma d'utilisation le plus efficace avec une charge utile importante et peu de ressources dépensées en *overhead*. Bien qu'utilisant une part importante de ses ressources, l'approche préemptive en dépense trop en *overhead*, et finalement, offre peu de charge utile.

☐ Les solutions distribuées arrivent à répartir la charge au sein du système à l'aide du mécanisme de transfert. À quelle fréquence et avec quelle efficacité les approches distribuées utilisent-elles le mécanisme de transfert ?

3.3.3 Qualité de la distribution

Transfert et communication

La figure 3.8 montre le nombre cumulatif de succès et d'échecs de transferts pour les 500 itérations. L'approche DQN s'avère être la plus efficace transférant davantage (de

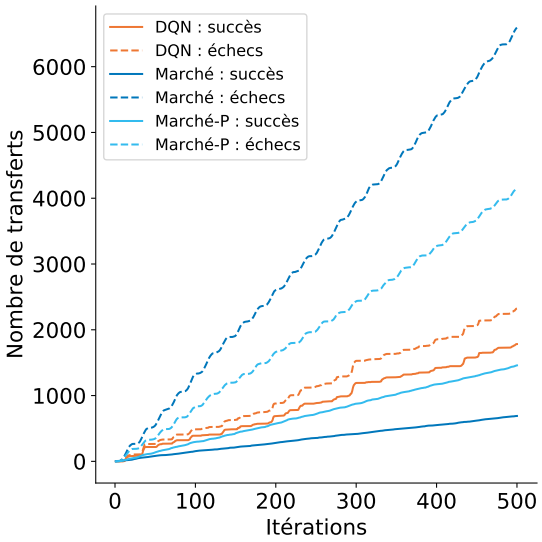

FIGURE 3.8 – Nombres cumulés de succès et d'échecs de transfert au cours des 500 itérations pour chaque solution distribuée.

trois à six fois plus que les deux autres approches avec un ratio succès/échecs nettement plus favorable. Or, comme noté précédemment, cette approche génère peu d'*overhead*. Par conséquent, nous constatons qu'une stratégie de transfert intensive des tâches peu couteuse est mise en place afin de limiter cet *overhead*. De plus, cela atteste de la capacité du système à conserver des ressources libres pour accepter d'éventuelles demandes de

transfert. Cependant, si comme souhaité notre système de récompense permet de contrôler le nombre de transferts, il échoue à éliminer totalement les échecs car cela ne dépend pas uniquement de l'agent transférant.

☐ La capacité de transfert dépend directement du nombre d'acceptants potentiels. Qu'arrive-t-il si nous étendons la portée du transfert à l'ensemble des zones ?

Incidence des zones

Les figurent 3.9 et 3.10 illustrent le nombre cumulatif de succès et d'échecs de transferts lorsque la portée est maximale et la comparaison des résultats selon la portée du transfert.

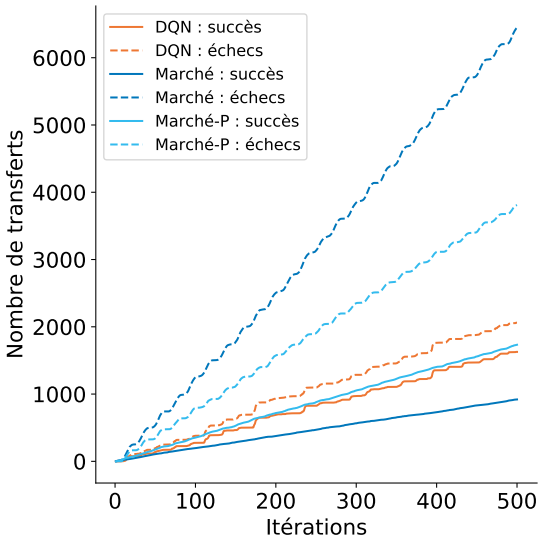

FIGURE 3.9 – Nombres cumulés de succès et d'échecs de transfert sans limitation de portée au cours des 500 itérations pour chaque solution distribuée.

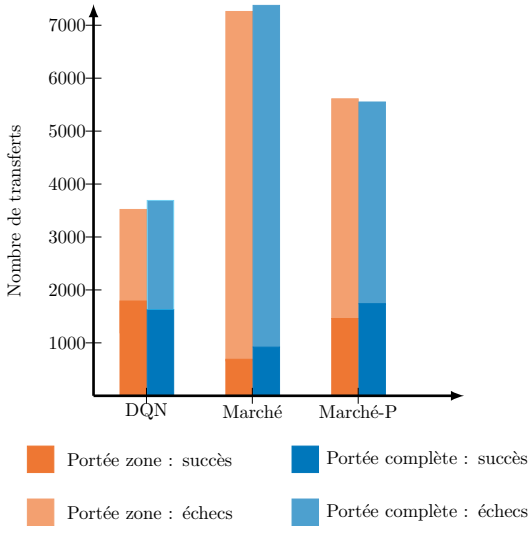

FIGURE 3.10 – Comparaison des succès et échecs des transferts en fonction de la portée du mécanisme.

Comme attendu, l'augmentation de la portée permet aux agents de trouver de nouvelles opportunités de vente. Cela se reflète dans les résultats obtenus par les approches basées sur le marché. En revanche, l'approche de type DQN ne profite pas de ce changement et, plus surprenant, diminue son nombre de transferts augmentant davantage son contrôle en limitant les échecs. Cette réticence à utiliser le système de transfert s'explique par l'*overhead* généré. En effet, cette solution transfère uniquement des tâches à haute priorité dont la complétion représente une plus grande pénalité que l'*overhead* qu'elle génère. Nous retrouvons le comportement observé précédemment où l'approche par DQN

ne transférait que des tâches à bas coût pour limiter l'*overhead*.

☐ Jusqu'à maintenant, la solution DQN a su démontrer ses capacités d'adaptation. Comment se comporte-t-elle à plus grande échelle ou en présence d'un déséquilibre ?

3.3.4 Évolutivité et déséquilibre

Évolutivité

Afin de tester l'évolutivité de nos solutions, nous augmentons la taille de notre système de 7 à 20 robots ainsi que le nombre moyen de tâches de 7 à 20. La gestion des zones reste identique. Si nous regardons le taux de complétion en figure 3.11, nous constatons des résultats similaires à ceux observés pour un système de 7 robots.

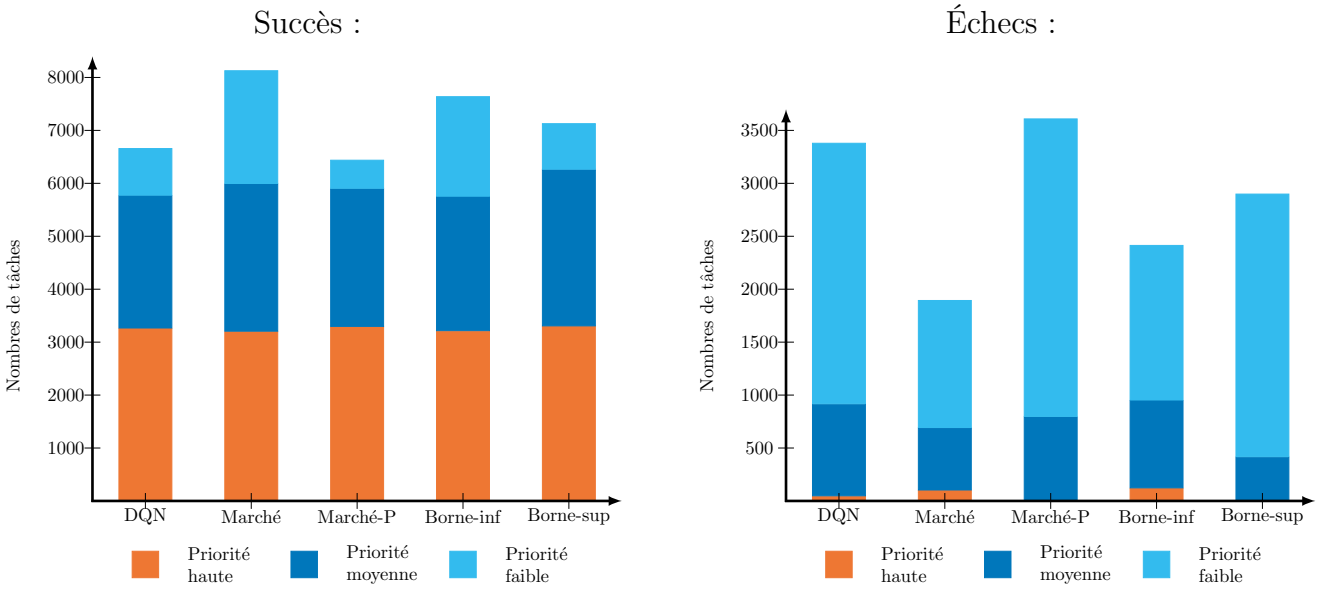

FIGURE 3.11 – Le graphique de gauche présente le nombre de tâches complétées par chaque solution utilisant 20 robots selon leur niveau de priorité. Celui de droite représente le nombre de tâches échouées.

La hiérarchie des performances d'utilisation des ressources est, elle aussi, similaire à celle constatée précédemment. Cependant, nous observons en figure 3.12 une légère augmentation globale (excepté pour la borne inférieure) des ressources utilisées due à la multiplication des possibilités d'allocation.

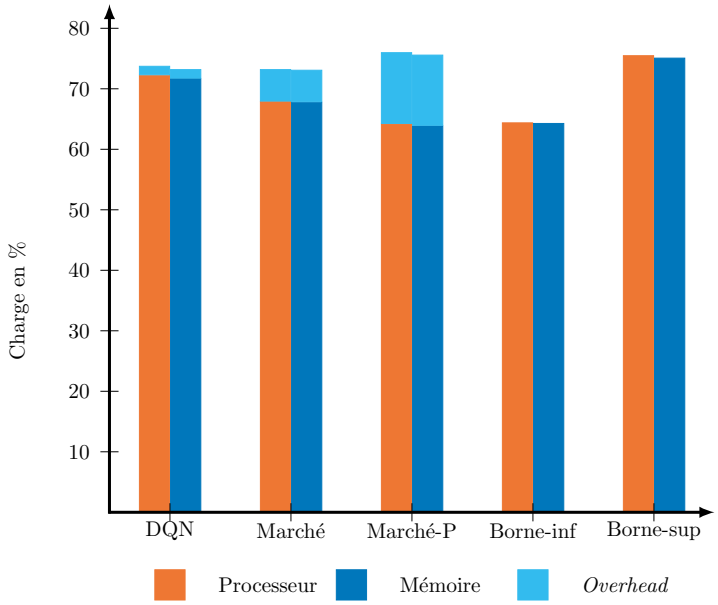

FIGURE 3.12 – Utilisation moyenne des ressources pour un système composé de 20 robots.

☐ Les solutions semblent insensibles au changement d'échelle, mais comment se comportent-elles en cas de prédominance d'une ressource calculatoire ?

Déséquilibre

Afin de générer un déséquilibre de l'importance des ressources calculatoires, nous introduisons un nouveau jeu de tâches présenté dans le tableau 3.5 où les tâches nécessitent en moyenne plus de CPU que de mémoire.

TABLE 3.5 – Jeu de tâches déséquilibré

Tâches								
ID	CPU (%)	RAM (%)	ID	CPU (%)	RAM (%)	ID	CPU (%)	RAM (%)
1	5	5	6	30	30	11	30	5
2	10	10	7	35	35	12	30	20
3	15	15	8	40	40	13	35	20
4	20	20	9	15	10	14	35	25
5	25	25	10	25	15	15	40	10

Les taux de complétion et la gestion des priorités exposés en figure 3.13 s'avèrent proches de ceux observés avec un jeu de tâches équilibré, mais l'utilisation des ressources diffère significativement.

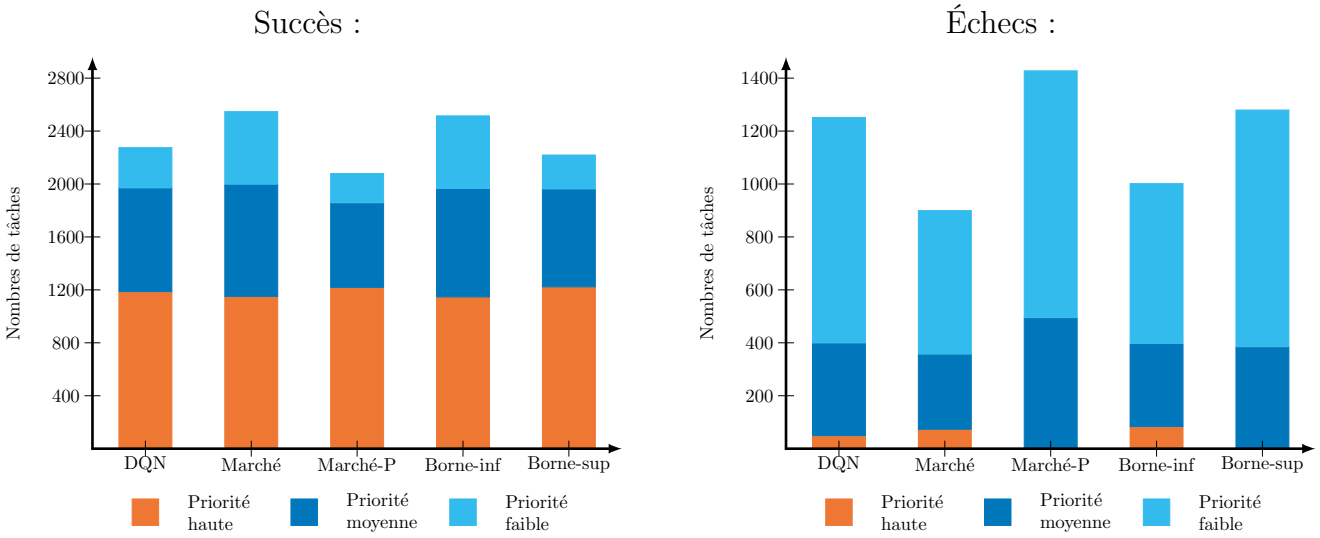

FIGURE 3.13 – Le graphique de gauche présente le nombre de tâches complétées par chaque solution selon leur niveau de priorité lors de l'utilisation d'un jeu de tâches déséquilibré. Celui de droite représente le nombre de tâches échouées.

En effet, comme constaté en figure 3.14, l'approche utilisant un DQN parvient à surpasser les performances de la borne supérieure. Cela reflète une profonde différence entre les approches étudiées. D'un côté, nous observons des approches classiques (sans apprentissage par renforcement) limitées par leur nature statique. Elles sont définies pour un problème donné et ne peuvent s'adapter. Le processus de valorisation d'une offre fournit un bon exemple. En effet, il confère une importance égale aux deux ressources. Or, dans le cas présent, la ressource CPU possède plus de valeur que la ressource mémoire. De l'autre côté, l'approche par DQN démontre son caractère adaptatif et apprend, par elle-même, la véritable valeur d'une ressource. Il en résulte un système fonctionnel et performant ne nécessitant pas d'informations *a priori*.

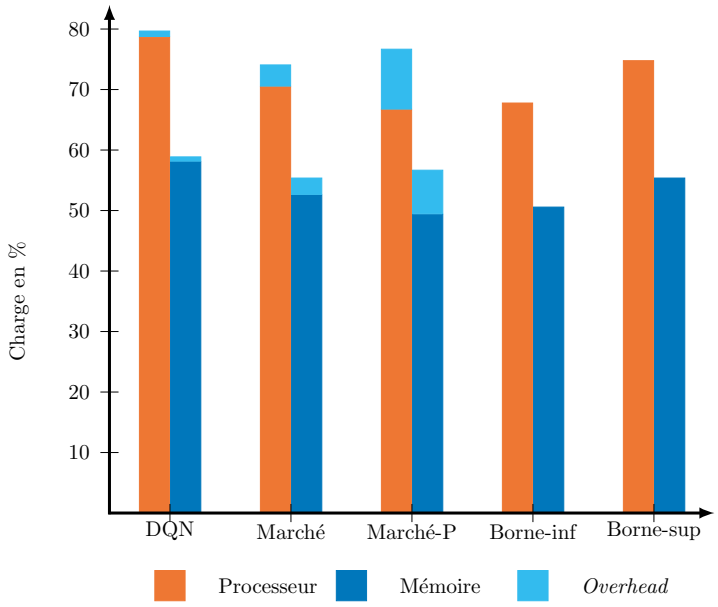

FIGURE 3.14 – Taux moyen d'utilisation des ressources lorsque le second jeu de tâches est utilisé.

3.4 Conclusion

Dans un contexte totalement distribué, des agents doivent se répartir la charge de calcul à l'aide d'un mécanisme de transfert afin d'éviter les surcharges locales. Afin de complexifier le problème, nous avons instauré plusieurs limitations à ce mécanisme : la génération d'un *overhead* représentant 20% des ressources nécessaires à l'exécution de la tâche ainsi qu'une portée maximale.

En raison de la complexité croissante des systèmes modernes, nous avons imposé un système de valorisation des offres simple afin de représenter l'impossibilité d'évaluer correctement une offre dans un système réel. De même, nous avons considérablement limité la taille de l'architecture DQN afin de la rendre compatible avec les capacités de calcul d'un robot mobile. De plus, comme pour la valorisation des offres, nous avons imposé un système de récompense simple. Bien que diminuée par ces contraintes, l'approche utilisant un DQN, qui apprend sans connaissance initiale, s'est montrée pertinente en étant capable de :

1. Respecter une hiérarchie (non imposée) des priorités en privilégiant les tâches à forte priorité.
2. Ordonnancer efficacement des tâches afin de maximiser l'utilisation des ressources disponibles.
3. Développer une stratégie de transfert en favorisant les tâches à bas coût et haute priorité, limitant ainsi l'*overhead* généré.
4. S'adapter à un environnement dynamique en valorisant les ressources calculatoires lorsqu'un jeu de tâche déséquilibré est utilisé.

Cette approche s'avère donc capable de générer son propre modèle de l'environnement pertinent sans nécessiter de connaissance préalable, et par conséquent, s'affranchit d'une contrainte forte. Cependant, elle comporte deux limitations attendues :

1. Bien que respectant la hiérarchie des priorités, cette solution s'avère incapable de garantir l'exécution des tâches à haute priorité.
2. De plus, il nous a été impossible de réduire suffisamment le nombre d'échecs de transferts. Cela s'explique par le caractère totalement distribué du système qui ne permet pas de mettre en place une récompense d'équipe pour "coordonner" l'apprentissage/prise de décision.

3.5 Note importante

Ce chapitre a fait l'objet d'une publication à IRC'20 qui a été sélectionnée pour une version étendue soumise en revue. De nouveaux résultats ont été produits afin d'explorer un certain nombre de paramètres (ex. génération de tâches). Ces derniers, ayant été obtenus tardivement, ne sont pas présentés.

Chapitre 4

Administration dynamique d'un cluster robotique

Sommaire

4.1 Introduction

Dans le chapitre précédent, nous avons abordé la question de la prise en considération de la charge calculatoire dans un contexte de MRTA avec la mise en place d'un mécanisme de transfert au sein d'un cluster robotique. Cependant, ces clusters sont avant tout conçus pour paralléliser des tâches de calculs intensifs qu'un robot seul peinerait à exécuter, telles que la fusion de capteurs ou les méthodes de vision par ordinateur utilisant de l'apprentissage profond.

Or contrairement aux ressources d'un cloud robotique pouvant être considérées comme illimitées, les ressources d'un cluster s'avèrent limitées et fluctuantes car dépendantes des variations de charges subies par le système. De plus, elles s'utilisent au détriment des autres membres du cluster. Par conséquent, il apparaît nécessaire d'instaurer un mécanisme d'allocation des ressources distribuées afin de favoriser les tâches prioritaires, de prédire les disponibilités et de maintenir un mécanisme réactif.

Contenu de la nature dynamique du problème, une solution basée sur de l'apprentissage pourrait jouer ce rôle de gestionnaire des ressources du cluster. De plus, afin de pouvoir illustrer ce dynamisme, nous ajoutons un contexte opérationnel au problème.

Les missions de recherche et sauvetage (S&R) illustrent parfaitement cette nécessité pour les systèmes multi-robots de pouvoir exécuter localement ces tâches calculatoires. En effet, d'une part ces missions S&R prennent place dans des milieux où les infrastructures de communications sont endommagées ou inexistantes rendant l'accès à des serveurs extérieurs erratique voire impraticable. D'autre part, la nature dynamique et incertaine de ces missions exige rapidité et adaptabilité afin de pouvoir compenser le manque d'informations. Or, le recours au cloud robotique génère une importante latence peu adapté à ce niveau de criticité.

Dans ce chapitre, nous explorons l'utilisation de méthodes d'apprentissage par renforcement (compatibles avec les espaces d'action à fortes dimensions) pour organiser la parallélisation des tâches au sein d'un cluster robotique déployé dans le cadre d'une mission de S&R en répondant aux questions suivantes :

1. Dans quelle mesure la parallélisation des tâches peut-elle améliorer les performances d'un système multi-robots ?

2. Les robots d'un système totalement décentralisé peuvent-ils apprendre à organiser la parallélisation des tâches ?

4.2 Présentation du problème

De nombreux concepts étant imbriqués, nous allons les présenter rapidement avant de décrire leurs modélisations.

4.2.1 Les concepts principaux

Certains des concepts décrits s'avèrent proches de ceux étudiés au chapitre 3, mais de nouveaux interviennent comme la consommation d'énergie, ou encore, l'échec possible de la mission.

Les objectifs de la mission

L'objectif principal du système est de conduire une mission de recherche et sauvetage avant qu'un de ses drones ne tombe à court d'énergie. Il existe donc une notion de succès ou d'échec. Le deuxième objectif du système consiste à terminer (avec succès), le plus rapidement cette mission. La difficulté d'une mission dépend des caractéristiques de l'aire.

L'aire de la mission

Chaque mission se déroule dans une aire spécifique dont les caractéristiques, exposées ci-dessous, définissent la spécificité :

1. Le danger reflète les conditions environnementales dans lesquelles les drones évoluent et influe directement sur l'efficacité de certains types de tâches.
2. La densité de région d'intérêt (ROI) représente la concentration potentielle de personne à secourir, et comme le danger, influence l'effet de certains types de tâches.
3. Le facteur de planification de trajectoire définit la vitesse d'accroissement du besoin de planification. Autrement dit, plus ce facteur est élevé, et plus rapidement, une nouvelle trajectoire, devra être recalculée.
4. Le facteur de fusion de données, analogiquement au facteur précédent, fixe l'importance de la fusion de données (synchronisation), c'est à dire la vitesse d'augmentation de ce besoin.

5. Les paliers de sauvetages définissent la fréquence à laquelle les tâches de secours sont activées par les drones.

Une aire est donc une entité abstraite dont les spécificités influent sur la capacité du système multi-robots.

Le système multi-robots

Afin de répondre aux exigences d'un déploiement dans un milieu critique tel qu'une zone de sinistre, le système multi-robots est décentralisé et isolé de tout appui extérieur (cloud robotique). En outre, bien que capable de communiquer entre eux, les drones du système évoluent indépendamment sans échanger d'informations concernant leur état. Par conséquent, les communications apparaissent uniquement lors de la distribution d'une tâche.

Une nouvelle fois, nous nous intéressons aux tâches d'un point de vue calculatoire. Dès lors, la définition de nos drones s'avère proche de celle utilisée précédemment. Néanmoins, dans un contexte opérationnel réel, un système de drones exécute une mission sous une contrainte forte, l'énergie initiale dont il dispose. Cette dernière sera consommée en fonction des choix et des tâches réalisés par le système. Pour cette raison, nous considérons un modèle de consommation afin d'ajouter cette contrainte forte au problème posé.

La consommation d'énergie

L'ajout d'une notion énergétique à la mission complexifie le problème étudié : il ne s'agit plus seulement de distribuer les tâches afin de maximiser l'utilisation des ressources de calcul, mais de distribuer les tâches afin de maximiser l'utilisation des ressources de calculs tout en répartissant les tâches énergivores pour s'assurer qu'aucun drone ne tombe à court d'énergie.

En vue d'intégrer cette notion, nous modélisons la consommation de chaque drone selon deux axes. Le premier correspond à la dépense énergétique induite par les déplacements du drone. Nous partons du principe que cette dernière augmente exponentiellement avec la vitesse du drone [4]. Le deuxième axe correspond à la consommation des composants électroniques embarqués sur le robot (processeur, GPU, capteurs, Wifi) qui croit selon la charge calculatoire (ajustement dynamique du couple tension/fréquence).

Or, la consommation d'énergie d'un drone dépend fortement des tâches qu'il exécute.

4.2.2 Le concept de tâche

Nous avons fortement augmenté la complexité de notre modèle de tâche afin, d'une part, d'intégrer un mécanisme de parallélisation et, d'autre part, de lui concéder, lors de sa complétion, un effet sur l'environnement. Cet effet dépend du type de la tâche.

Types

Notre modélisation considère cinq types de tâches réparties en deux catégories. La première catégorie regroupe les tâches participant directement à l'avancement de la mission. Elle est constituée des types : 1) exploration, 2) recherche et 3) secours qui possèdent une chaine de dépendance. En effet, il est nécessaire d'avoir préalablement conduit des tâches d'exploration pour mener des tâches de recherche, et de même, il est obligatoire de rechercher avant de secourir. La deuxième catégorie composée des types 4) planification de trajectoire et 5) fusion de données dont les effets respectifs réduisent les risques de collisions et la désynchronisation. Bien que n'influant pas directement sur la complétion de la mission, la non-exécution de ces types diminue l'efficacité des tâches appartenant à la première catégorie allant jusqu'à les rendre inopérantes. Chacun des types mentionnés renvoie aux applications réelles suivantes :

1. Exploration : cartographie de l'aire à l'aide d'un LIDAR afin de définir la topographie du terrain et d'identifier les zones critiques nécessitant des recherches avancées.
2. Recherche : examine attentivement par traitement d'image les sections critiques afin de découvrir les cibles nécessitant assistance.
3. Sauvetage : tourne autour de la cible afin d'évaluer la situation à l'aide de méthodes de diagnostique multi-capteurs et calcule l'action à entreprendre au moyen de techniques de planification de mission
4. Planification de trajectoire : recalcule la trajectoire des drones afin d'éviter les collisions et d'améliorer l'efficacité du système.
5. Fusion de données : fusionne les données collectées afin d'obtenir une meilleure évaluation de la situation et de prévenir le chevauchement des recherches.

Le type d'une tâche conditionne les pré-requis nécessaires à son exécution/distribution et est fixé au moment de sa création.

Génération des tâches

Nous estimons que les drones d'un système distribué devraient être les plus autonomes possible afin d'accélérer le processus décisionnel et de limiter les communications. Par conséquent, chaque drone tente de commencer l'exécution de tâche distribuables sans consultation préalable en s'appuyant uniquement sur leur compréhension de l'état de la mission. Le choix du type de tâches dépend alors de probabilités dynamiques évoluant selon la compréhension du drone.

Paradigme de distribution

Afin de permettre la distribution/parallélisation d'une tâche, nous avons affiné le concept de tâche en intégrant le concept de *job* selon un fonctionnement serveur/travailleurs (ou maître/esclaves comme décrit dans [3]). Ainsi une tâche est divisée en un job de type serveur et i jobs de type travailleur avec $1 \leqslant i \leqslant n$ où n représente le degré maximal de parallélisation de la tâche. Nous avons inclus deux paradigmes de distribution présentés en Figure 4.1 qui diffèrent par la localisation des données.

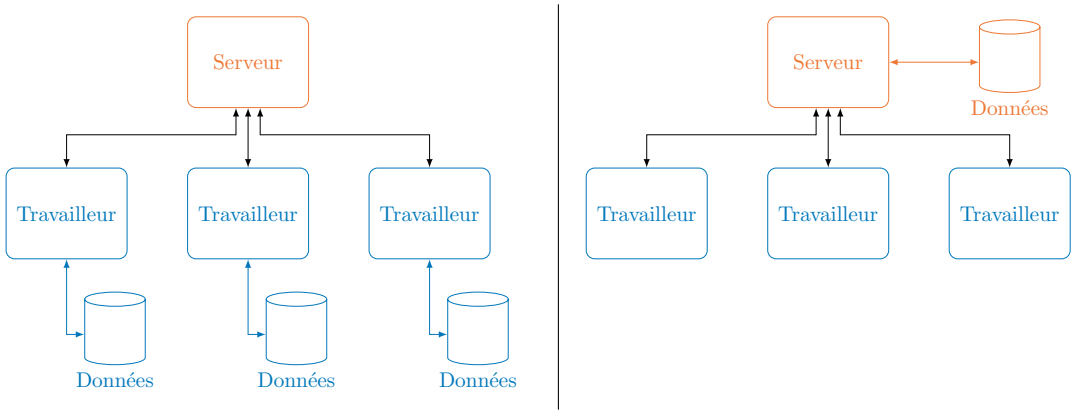

FIGURE 4.1 – Deux paradigmes de distribution : à gauche, les travailleurs possèdent les données nécessaires à l'exécution de la tâche alors qu'à droite le serveur doit transmettre les données les données aux travailleurs.

Concernant les effets de la parallélisation, deux approches existent. La première considère qu'une tâche parallélisée est accélérée ce qui permet de terminer son exécution plus rapidement. La seconde approche considère que la parallélisation d'une tâche n'accélère pas son exécution, mais augmente le nombre de données traitées, et donc, l'efficacité (l'ampleur de l'effet) de la tâche. Bien que plus répandue, la première approche s'avère

impraticable pour nos simulations. En effet, le nombre de cycles nécessaires à l'exécution d'une tâche se doit d'être entier (cf. chapitre 3). Si nous suivons la loi d'Amdahl dont la formule est la suivante :

$$S_{\text{latence}} = \frac{1}{1 - p + \frac{p}{s}} \tag{4.1}$$

Un temps d'exécution C compatible avec une simulation discrète nécessite que : $\forall s \in [\![1, n]\!]$, où n représente le nombre maximal de travailleurs et p la partie de la tâche pouvant profiter de l'accélération, les $\frac{C}{S(s)}$ sont des nombres entiers. Or, comme attesté dans le Tableau 4.1, le nombre de cycles devant être simulé est considérable. De plus, les pourcentages d'accélération compatibles devant être utilisés s'avèrent extrêmement faibles. Par exemple, il n'existe aucun entier compatible inférieur à 50000 pour un nombre de travailleurs maximal de cinq et une tâches parallélisable à 95%.

Table 4.1 – Premiers entiers compatibles en suivant la loi d'Amdhal

Premier entier compatible	**Nombre de cycles selon le nombre de travailleurs**	**Parallélisation maximale**	**% parallélisabe de la tâche**
80	80, 50, 40, 35, 32	5	75
120	120, 90, 80, 75, 72	5	50
560	560, 350, 280, 245, 224, 210, 200	7	75
1680	1680, 1470, 1400, 1365, 1344, 1330, 1320	7	25

Nous avons donc opté pour la deuxième approche où la parallélisation d'une tâche accroit le nombre de données traitées. Ainsi, une tâche accélérée nécessite autant de cycles pour être complétée, mais son effet est démultiplié par le nombre de travailleurs participants. Dans notre modèle, les travailleurs traitent les données et le serveur répartit le travail (et si besoin les données) et synchronise les résultats.

Nous avons présenté l'ensemble des concepts de notre problème et la mission qui lui est attachée. Nous allons maintenant détailler la modélisation de chacun de ces concepts.

4.3 Modélisation du problème

Les approches étudiées partagent toutes la même définition de l'environnement et des acteurs.

4.3.1 Modélisation de l'environnement et des acteurs

Modélisation de l'aire de la mission

Comme mentionné précédemment, une aire est définie par ses caractéristiques répertoriées dans le Tableau 4.2 et dont la valeur varie entre $0 \rightarrow 1$. Elles sont réparties en trois catégories :

1. Les caractéristiques d'**état** représentent le niveau de difficulté opérationnelle de l'aire et influent directement sur l'efficacité des tâches effectuées par le MRS.
2. Les caractéristiques de **progression** décrivent l'avancement de la mission. Lorsque R atteint 1, la mission est terminée avec succès. Leurs valeurs évoluent lorsque leurs tâches respectives sont complétées sachant que la chaîne de dépendance doit être respectée (Exploration $\longrightarrow$ Recherche $\longrightarrow$ Sauvetage possible).
3. Les caractéristiques de **contraintes** décrivent les besoins opérationnels du MRS et portent atteinte à l'efficacité des tâches nécessaires à l'avancement de la mission.

Table 4.2 – Liste des caractéristiques d'une aire

Caractéristiques	**Notations**	**Descriptions**	**Évolue**	**Types**
Danger	H	Conditions environnementales	Non	État
Densité ROI	D	Densité des zones d'intérêts	Non	État
Exploration	E	Niveau d'exploration courant	Oui	Progression
Recherche	R	Niveau de recherche	Oui	Progression
Sauvetage	S	Secours effectué	Oui	Progression
Sauvetage possible	Sp	Palier de secours débloqué	Oui par palier	État
Planification de trajectoire	P	Besoin de replanification	Oui	Contrainte
Fusion de données	M	Besoin de synchronisation	Oui	Contrainte
Facteur de planification	λ_P	Conditionne l'efficacité de la planification	Non	État
Facteur de fusion	λ_F	Affecte la fusion de données	Non	État

À chaque itération i, le besoin de planification est mis à jour en fonction du nombre de drones commençant un job nécessitant un déplacement selon la formule :

$$P_i = P_{i-1} + \lambda_P \text{Card}(\Omega_i) \tag{4.2}$$

où Ω_i représente l'ensemble des nouveaux jobs nécessitant un déplacement dont l'exécution a débuté à l'itération i

De la même manière que pour la planification, le besoin de fusion est actualisé à chaque itération i. Son augmentation dépend du nombre de tâches d'avancement réalisées selon la formule suivante :

$$M_i = M_{i-1}\lambda_F \text{Card}(\Gamma_i) \tag{4.3}$$

où Γ_i est l'ensemble des tâches d'avancement complétées à l'itération i.

Pour conclure, une aire représente l'environnement de la mission et notamment sa difficulté. Par conséquent, différentes aires de mission signifient différentes missions, il n'existe pas de lien entre les aires et l'ensemble du système multi-robots est déployé dans une unique aire au cours d'une mission. Autrement dit, une aire équivaut à un scénario de test.

Modélisation du système multi-robots

Le système multi-robots est composé de n drones déployés simultanément lors du début de la première itération. Les drones connaissent les caractéristiques courantes de l'aire (excepté λ_P et λ_F) ainsi que leurs propres caractéristiques résumées dans le tableau 4.3. En revanche, ils ne possèdent aucune information sur l'état des autres membres du système et les tâches en cours d'exécution. De plus, bien que pouvant posséder des capacités calculatoires ou énergétiques différentes, leur motorisation et habilité à conduire une tâche sont identiques.

Pour conclure, les drones du système sont définis sous les angles calculatoires et énergétiques. Leur localisation n'entre pas en considération dans notre modélisation. Les notions de déplacement et de vitesse servent à modéliser un facteur consommation énergétique lié à la motorisation des drones.

TABLE 4.3 – Tableau des caractéristiques principales d'un drone

Caractéristiques	Descriptions	Types
ID	ID du drone	Entier
CPU	Quantité de ressources CPU utilisée	Flottant
Mémoire	Quantité de mémoire utilisée	Flottant
Batterie	Quantité d'énergie restante	Flottant
Déplacement	Atteste du déplacement du drone	Booléen
Vitesse	Palier de vitesse actuel du drone	Entier
Jobs de travailleur	Liste des jobs de type travailleur exécutés par le drone	Liste
Jobs de serveur	Liste des jobs de type serveur exécutés par le drone	Liste

Modélisation du temps

Comme expliqué dans le chapitre précédent, il est nécessaire de discrétiser la temporalité. C'est à dire la diviser en itérations de durées équivalentes représentant un δ_t de temps continu comme illustré en figure 4.2. Cela entraîne les conséquences suivantes :

1. Les robots reçoivent simultanément leurs nouvelles tâches et les traitent en parallèle.
2. Plusieurs tâches commencent ou terminent exactement au même moment comme décrit en figure 4.2.b.

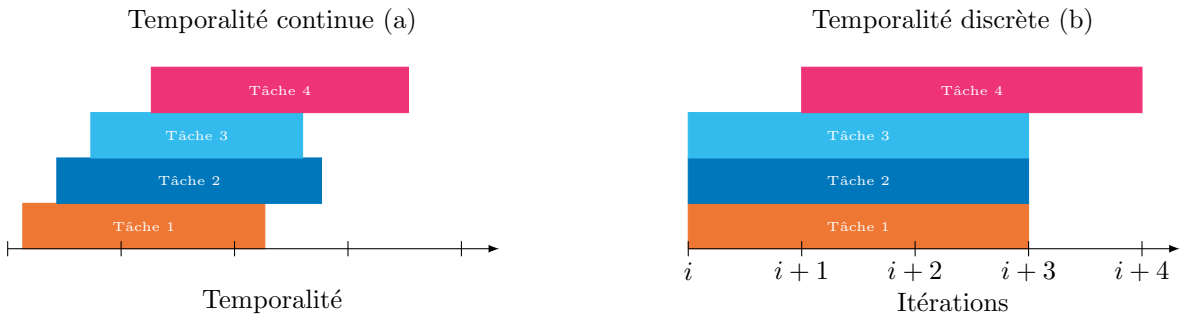

FIGURE 4.2 – Temporalité : contrairement à une temporalité continue, une temporalité discrète implique de la simultanéité.

Minimiser le nombre d'itérations nécessaires à la réalisation de la mission représente le deuxième objectif du système. Pour cela, le système doit exécuter de façon efficace une

succession de tâches.

4.3.2 Modélisation d'une tâche

Les tâches constituent l'élément clé de la mission car elles permettent sa progression. Afin d'intégrer le principe de parallélisation, nous avons affiné notre définition d'une tâche en la divisant en *jobs*.

Définition d'une tâche

Une tâche possède un type et se compose d'un job serveur et de n jobs travailleur ($n > 0$). Ainsi, contrairement au chapitre 3, un drone n'accepte pas d'exécuter une tâche, mais un ou plusieurs *jobs* d'une tâche. Les caractéristiques d'une tâche et d'un *job*, listées respectivement dans les tableaux 4.4 et 4.5, dépendent de leur type.

TABLE 4.4 – Caractéristiques d'une tâche

Caractéristiques	**Descriptions**	**Types**
ID	ID de la tâche	Entier
Type	Le type de la tâche	String
Nombre de travailleurs	Nombre maximum de travailleurs autorisés	Entier
Serveur	Le serveur de la tâche	Job
Travailleurs	La liste des jobs travailleurs de la tâche	Liste

Il n'existe aucun mécanisme de préemption, lorsqu'un *job* est commencé, il doit être terminé. De plus, la parallélisation est synchrone, par conséquent, tous les *jobs* d'une tâche commencent et terminent à la même itération.

Le type d'une tâche est sélectionné à sa création au moyen d'une fonction de probabilités dynamiques.

Sélection du type

Toutes les I itérations, $I \in \mathbb{N}$, chaque drone du système tente de démarrer une nouvelle tâche distribuée. Le type de cette tâche dépend de l'état du drone et de sa compréhension de l'état de la mission. La sélection s'effectue selon les probabilités décrites dans le tableau 4.6. Afin de modéliser le fait qu'un drone puisse ne pas se trouver dans une situation

TABLE 4.5 – Caractéristiques d'un job

Caractéristiques	Descriptions	Types	Concerne
ID	ID du Job	Entier	Serveur/travailleur
Robot	ID du robot exécutant le job	Serveur/travailleur	
Type	Le type du job	String	Serveur/travailleur
CPU	Quantité de CPU nécessaire à l'exécution du job	Flottant	Serveur/travailleur
Mémoire	Quantité de mémoire nécessaire à l'exécution du job	Flottant	Serveur/travailleur
Déplacement	Déplacement nécessaire	Booléen	Serveur/travailleur
Vitesse	Palier de vitesse nécessaire pour le déplacement	Entier	Serveur/travailleur

opportune pour débuter une tâche, il existe une probabilité pour qu'un drone ne démarre aucune tâche

TABLE 4.6 – Probabilités de sélection de chaque type de tâches

Types	Probabilités
Exploration	$P(T_E) = \begin{cases} 1 - E \\ \\ 0 \text{ si en déplacement} \end{cases}$
Recherche	$P(T_R) = E - R$
Sauvetage	$P(T_S) = \begin{cases} R - Sp \\ \\ 0 \text{ si en déplacement} \end{cases}$
Planification de trajectoire	$P(T_P) = M$
Fusion de donnée	$P(T_F) = F$
Aucune	$P(T_0) = \tau$

Une fois le type sélectionné, le drone exécute le job de type serveur de la tâche et transfère, au commissaire-priseur, autant d'offres de travailleurs que le type de la tâche en autorise afin de la distribuer. L'affectation des travailleurs dépend de la solution employée et déterminera fortement l'action de la tâche.

Effets des tâches

Chaque type de tâche influe sur la caractéristique correspondante de l'aire de la mission (ex : une tâche de type "exploration" accroit le niveau d'exploration de l'aire). L'ampleur de la modification dépend du nombre de travailleurs, de la pénalité liée aux contraintes et de la restriction imposée par la chaîne de dépendances. Les tâches, directement nécessaires à l'avancement, commencées à l'itération i subissent lors de leur complétion une pénalité d'efficacité $\mathcal{P}_i$ liée à l'absence de replanification (P_i^2) et au manque de synchronisation (F_i^2) dont la valeur est calculée selon la formule suivante :

$$\mathcal{P}_i = (1 - \mathrm{P}_i^2)(1 - \mathrm{F}_i^2) \tag{4.4}$$

Le tableau 4.7 présente les effets de chaque tâche selon leur type en fonction de i_s, l'itération de commencement de la tâche, i_c, l'itération courante, ω, le nombre de jobs de type travailleur que possédait la tâche, μ_1 et μ_2 les gains respectifs des tâches de première et deuxième catégories.

TABLE 4.7 – Tableau des effets de chaque type : le nombre de travailleurs démultiplie les effets d'une tâche alors que les besoins les atténuent (pour les tâches de première catégorie).

Types	**Effets**
Exploration	$E = \mathrm{Min}\left(\frac{E_{i_c} + \mu_1 \cdot w \times \mathcal{P}_i}{H}, 1\right)$
Recherche	$S = \mathrm{Min}\left(\frac{S_{i_c} + \mu_1 \cdot w \times \mathcal{P}_i}{D}, E_{i_s}\right)$
Sauvetage	$R = \mathrm{Min}\left(\frac{R_{i_c} + \mu_1 \cdot w \times \mathcal{P}_i}{H + D}, Rp_{i_s}\right)$
Planification de trajectoire	$\mathrm{Max}(Np - \mu_2 \times w, 0)$
Fusion de données	$\mathrm{Max}(Nm - \mu_2 \times w, 0)$

La complétion de tâches n'implique pas uniquement la progression de la mission. En effet, leur exécution nécessite l'utilisation de ressources énergétiques, et donc, diminue la batterie des exécutants.

4.3.3 Modélisation de la consommation d'énergie

Le niveau de batterie des drones représente la contrainte principale de notre problème puisque la mission s'arrête dès qu'une des entités du système arrive à court d'énergie. Afin de pouvoir intégrer cette contrainte à notre problème, nous avons conçu un modèle de consommation considérant deux vecteurs de consommation la propulsion et les composants électroniques.

Consommation des composants électroniques

Dans cette catégorie, nous considérons deux sources de consommation. La principale provient des unités de calculs (SoC (Système sur puce) multi-cœurs, GPU et Mémoire) dont la consommation dépend directement de la charge calculatoire. La deuxième correspond à la consommation liée aux communications.

Il s'avère extrêmement complexe d'obtenir une modélisation précise de la consommation d'un SoC moderne à l'aide de méthodes analytiques. En effet, la complexité du problème conduit, en pratique, à l'utilisation de modèles probabilistes. Ainsi, une solution consiste à générer un modèle haut niveau à l'aide de mesures. Cela a été fait dans [58] pour différents SOC hétérogènes récents correspondant à ceux pouvant être utilisées sur des drones.

Nous avons donc considéré cette approche afin de disposer d'un modèle à la fois réaliste et exploitable. De plus, nous avons également retenu une politique simple de gestion de l'énergie et des ressources reposant sur une activation progressive des cœurs conditionnée par l'atteinte de la fréquence maximale. Ce modèle s'avère évolutif et peut être ajusté pour des SoC et GPU différents dont les mesures sont connues. Le modèle de consommation ainsi obtenu est illustré en Figure 4.3.

Nous en déduisons la formule de consommation (en mW) suivante en fonction de la charge de calcul (en pourcentage), ρ :

$$C_E(\rho) = 0.68\rho^2 + 42.62\rho + 1485.93 \tag{4.5}$$

Nous avons ajouté à cette consommation des processeurs, celle du Wifi des drones qui comprend un partie résiduelle représentant les communications limitées mais nécessaires

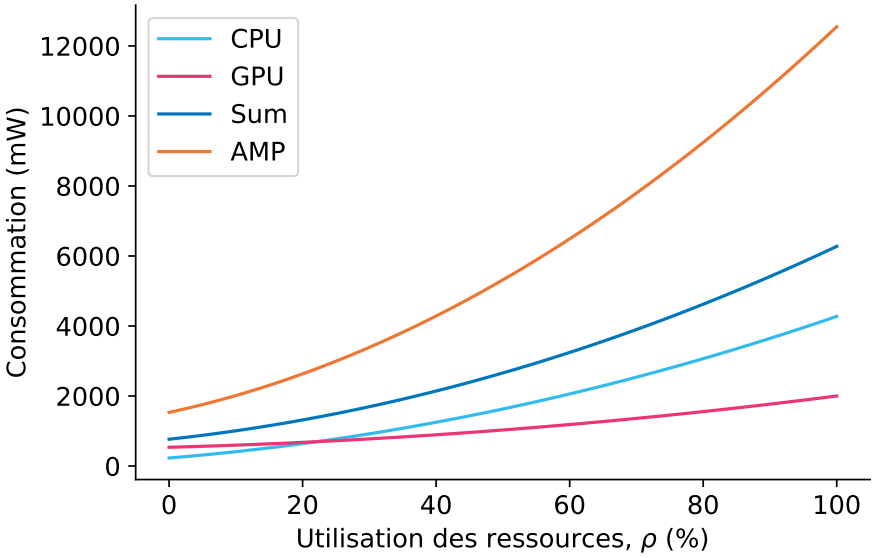

FIGURE 4.3 – Consommation d'un processeur mobile en fonction de l'utilisation. La courbe cyan représente la consommation du CPU. La magenta celle du GPU. Leur agrégation permet d'obtenir la courbe bleue qui peut être amplifiée par un facteur 2 afin de produire la courbe orange proche de la consommation d'une Jetson TX2.

entre les drones et une partie dépendant du nombre de jobs ω exécutés par le drone. Nous utilisons la formule suivante :

$$C_W(\omega) = 2000 + 1000\text{Card}(\omega) \tag{4.6}$$

La formule finale de la consommation de puissance moyenne liée aux composants correspond à l'agrégation des deux facteurs de consommation précédents :

$$C_H(\rho, \omega) = 0.68\rho^2 + 42.62\rho + 3485.93 + 1000\text{Card}(\omega) \tag{4.7}$$

Consommation liée à la propulsion

Pour cette partir, nous basons notre modèle sur le drone de surface Heron développé par la société CLEARPATH [1] et présenté ci-dessous :

FIGURE 4.4 – Photo du drone Heron

À partir des informations fournies par le constructeur [4], nous en avons déduit un modèle de consommation du drone en fonction de sa vitesse ν :

$$C_E(\nu) = 22.918\mathrm{e}^{1.127\nu} \tag{4.8}$$

De ce modèle, nous utilisons deux paliers consommant respectivement 20 W et 70 W comme illustré en figure 4.5. Nous considérons un drone en mouvement en situation de palier 1 et en situation de palier 0 sinon.

1. il s'agit d'un drone marin couramment utilisé dont le modèle 3D, les caractéristiques mécaniques ainsi que l'autopilote sont rendus disponibles.

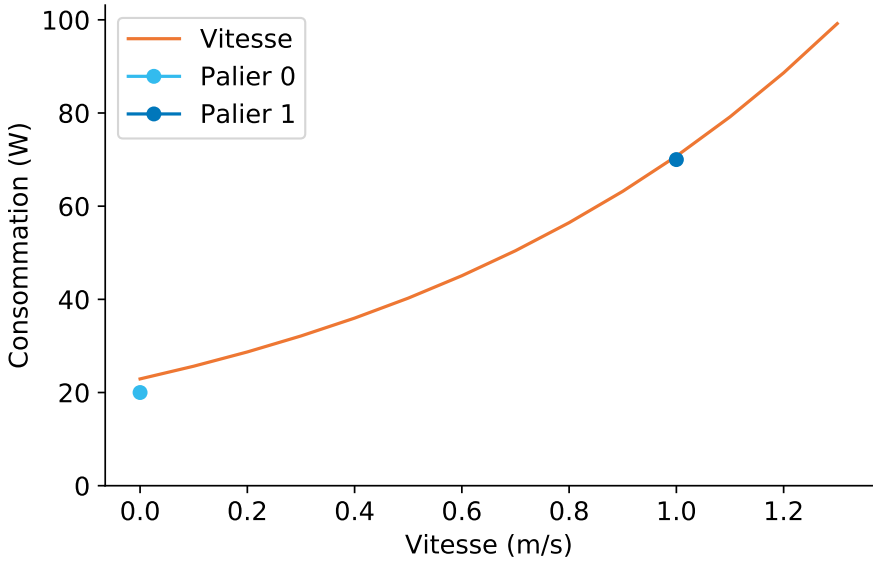

FIGURE 4.5 – Consommation liée à la propulsion en fonction de la vitesse du drone. Les paliers 0 et 1 apparaissent respectivement en cyan et bleu sur le graphique.

La gestion de l'énergie conditionne la réussite de la mission et figure donc au cœur des solutions que nous proposons

4.4 Définition de nos solutions

Le système ne possède pas de contrôle sur les tâches sélectionnées par les drones, mais il est responsable de l'allocation des *jobs* de type travailleur. Il peut donc indirectement favoriser certains types, se refréner d'allouer les ressources du système et réduire l'action de certains de ses membres.

4.4.1 Première solution

Toutes les solutions présentées s'appuient sur un mécanisme d'enchères doubles décrit en Section 2.2.1. Par conséquent, le déroulement des enchères est commun à toutes les solutions, seul le mécanisme d'attribution diffère.

Déroulement des enchères doubles

Lorsqu'un drone décide d'exécuter une nouvelle tâche, il démarre le *job* serveur correspondant et formule autant de demandes de travailleurs que le type en autorise comme illustré en figure 4.6. Ces offres de demandes possèdent deux valeurs, la première reflète le coût calculatoire du *job* et la seconde le niveau de batterie du drone effectuant la demande. Ainsi contrairement au chapitre 3, le choix de l'allocation s'effectue selon deux axes.

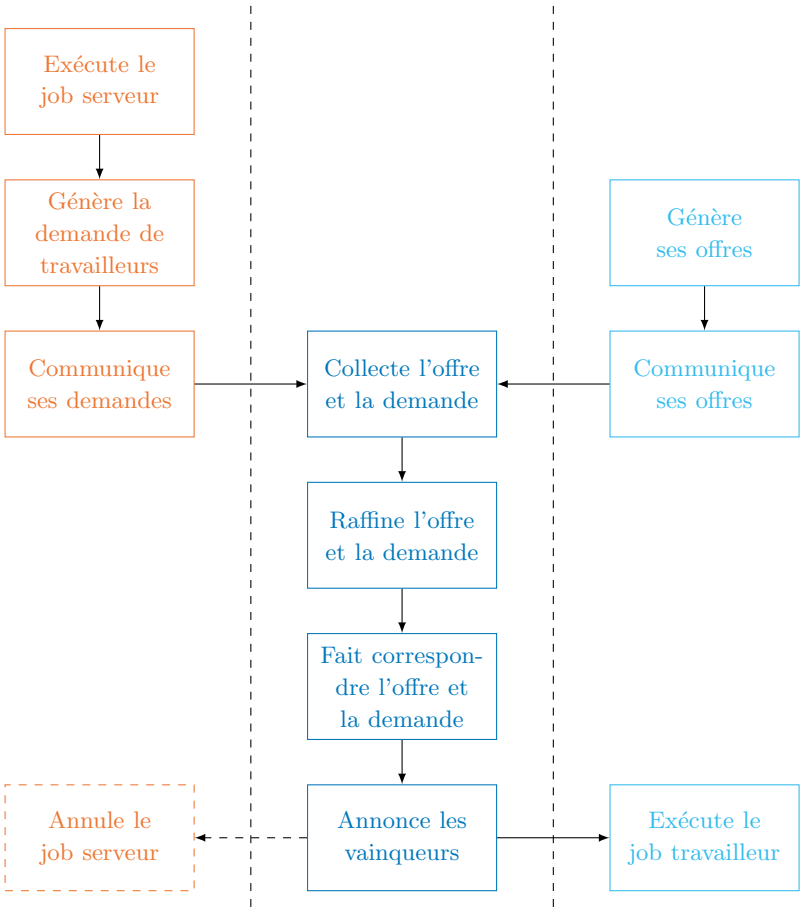

FIGURE 4.6 – Déroulement des enchères : chaque drone commençant une nouvelle tâche (en orange) génère une offre par travailleur potentiel. En parallèle, l'ensemble des drones (en cyan) formulent leurs offres d'achat. L'ensemble des propositions (offre et demande) est collecté par le commissaire-priseur (en bleu) qui conduit les enchères. À la fin, les gagnants exécutent leurs nouveaux *jobs* de travailleurs et les tâches ne trouvant pas suffisamment de travailleurs sont annulées, leur *job* serveur s'arrêtant.

En parallèle chaque drone du système soumet autant d'offres que ses capacités le permettent en suivant le procédé décrit dans l'algorithme 2. Comme pour les offres de demande, ces offres possèdent deux valeurs.

Algorithm 2: Multi-Bid valuation algorithm

Data:
L_{jobs}, the list of possible worker jobs
D_{cpu}, D_{mem}, the amount of free CPU and memory on the Drone respectively
D_{mov}, is true if the drone is currently moving
Function :
$Pro(i)$, $Mem(i)$, returns the processing and memory required by job i respectively
$Move(i)$, return True if the job i required to move
Result: *bids*, The bids list
bids = []
CPU, MEM, MOVE = D_{cpu}, D_{mem}, D_{mov}
while *True* **do**
  for j *in* L_{jobs} **do**
    tmp = False jobs = []
    if $CPU \geqslant Pro(j)$ *and* $MEM \geqslant Mem(j)$ **then**
      if $MOVE$ *and* $Move(j)$ **then**
        pass
      else if *not* $MOVE$ *and* $Move(j)$ **then**
        tmp = True jobs += [j]
      else
        jobs += [j]
  end
  if *jobs != []* **then**
    bids += [{
      'price' : $R_{CPU} + R_{MEM}$,
      'energy' : D_E,
      'jobs' : jobs
    }]
    if *move* **then**
      MOVE = tmp
    CPU -= Max([$Pro(i)$ for i in T]
    MEM -= Max([$Mem(i)$ for i in T]
  else
    break
end

Le commissaire-priseur [2] est ensuite chargé de conduire les enchères. Classiquement, les enchères doubles sont utilisées dans des milieux non-coopératifs où le rôle du commissaire-priseur est de fixer un prix qui satisfait les deux côtés du marché. Dans notre cas, le système est totalement coopératif, par conséquent, le commissaire-priseur tente de répartir l'offre et la demande de la façon qu'il juge la plus satisfaisante pour la mission. Nous proposons une première solution classique où le commissaire-priseur conduit les enchères de façon à allouer le maximum de travailleurs possibles par tâche (afin d'obtenir l'efficacité maximale) tout en privilégiant les forts niveaux de batteries. Indépendamment de la solution utilisée, les tâches ne trouvant pas plus de deux travailleurs sont automatiquement annulées, la distribution n'étant pas jugée suffisamment rentable.

Possibilités offertes par les enchères

Contrairement au travail précédent, l'offre et la demande sont collectées indépendamment sans nécessiter de communiquer les propositions de *jobs*. Cela est rendu possible par la standardisation des jobs au sein du cluster robotique (par exemple : un job de travailleur pour une tâche d'exploration nécessitera toujours les mêmes ressources). De plus, l'ensemble de l'offre et de la demande est attribué simultanément. Ce recours au mécanisme d'enchères doubles et à la standardisation offrent les avantages suivants :

1. Réductions des communications car la collecte des offres ne nécessite pas de communiquer des informations.
2. Mécanisme d'allocation efficace qui ne nécessite pas de concertation entre le serveur et ses futurs travailleurs.
3. Accélération du processus puisqu'un seul tour est nécessaire pour allouer l'ensemble de l'offre et de la demande.
4. Allocation selon deux axes (ressources de calcul et ressources énergétiques) afin de favoriser les drones en sous-charge ou/et peu exploités.
5. Mise en concurrence de l'offre et de la demande par la globalisation de la vision du commissaire priseur permettant de disqualifier les offres ou demandes jugées insatisfaisantes.

2. Contrairement au travail précédent, les enchères sont doubles par conséquent, toutes les demandes doivent être envoyées au même commissaire-priseur. Cependant, cela n'implique par la création d'un point de défaillance unique puisque n'importe quel drone peut jouer ce rôle.

Le raffinement des enchères offre au système décentralisé un certain contrôle sur le mécanisme autonome de sélection des tâches. Cependant, il reste limité puisque le commissaire-priseur ne peut forcer l'exécution d'une tâche et son application nécessite une forte compréhension de la situation. Devant ce constat, nous proposons deux autres approches basées sur l'apprentissage par renforcement pour décider du raffinement à opérer.

4.4.2 Approche BDQ

Principe

Notre objectif est de permettre à l'intelligence de décider de la pertinence des différentes tâches pour un état précis du système. Pour cela, lors de chaque enchère, nous lui demandons de choisir combien elle souhaite conserver de tâches pour chaque type. De plus, nous souhaitons qu'elle agisse aussi sur le plan énergétique en lui permettant de rejeter les offres et demandes possédant les valeurs d'énergie les plus faibles. Finalement, nous obtenons un espace d'action à sept dimensions où cinq dimensions sont liées aux types (une par type) et deux à la gestion de l'énergie (offre et demande). Or, nous avons défini par essais successifs les nombres suivants d'actions possibles (l'action détermine le nombre de tâches autorisées) : 7, 7, 9, 6, 9, 7, 7 pour les dimensions respectives : planification, fusion, exploration, recherche, sauvetage, offre et demande. Le nombre total d'actions à considérer est donc de $\prod_{d=1}^{N} n_d = 1166886$ actions. Nous avons donc opté pour une architecture BDQ où une branche sera dédiée à chaque sous-dimension

Architecture

Notre architecture, illustrée en figure 4.7, est constituée d'un réseau commun composé de quatre couches de 256 neurones chacune duquel émerge sept branches calculant les *advantages values* des sous-dimensions possédant chacune deux couches de respectivement 128 et 64 neurones et une autre estimant les *states-values* pourvue, elle aussi, de deux couches, l'une de 128 neurones et l'autre de 64. Comme expliqué dans le chapitre 2.1.3, les agrégations de chacune des branches *advantages values* avec la branche des *states-values* permet d'obtenir les *Q-values* de chaque sous-dimension de l'espace d'action. Il suffit ensuite de combiner la meilleure action dans chaque sous-dimension pour obtenir l'action finale.

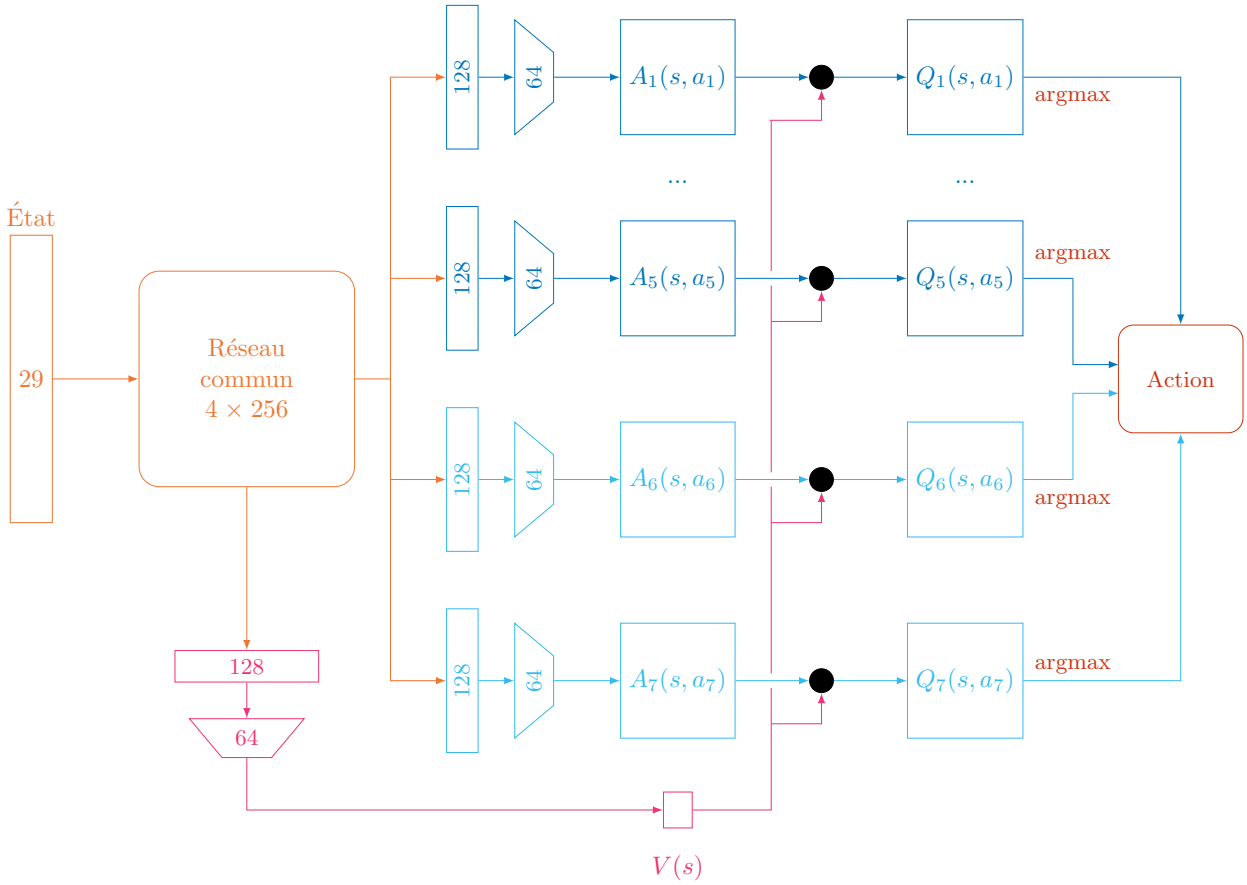

FIGURE 4.7 – Architecture BDQ : le réseau commun (en orange) est partagé entre les huit branches. La partie bleue représente les branches dédiées à l'estimation des *Q-values* des actions liées aux types. La partie cyan traite les actions relatives à la gestion de l'énergie. La branche en magenta calcule la *state-value*. À la fin, la sélection des meilleures actions de chaque branche permet de former l'action finale.

Nous utilisons la formule suivante pour réaliser l'agrégation :

$$Q_d(s, a_d) = V(s) + (A_d(s, a_d) - \frac{1}{n} \sum_{a'_d \in \mathcal{A}_d} A_d(s, a'_d)) \tag{4.9}$$

et celle-ci pour générer la cible TD :

$$y = r + \gamma \frac{1}{N} \sum_d Q_d^-(s', \arg\max_{a'_d \in \mathcal{A}_d} Q_d(s', a'_d)) \tag{4.10}$$

Système de récompenses

Nous avons exploré deux schémas de récompense différents. Le premier récompense différemment selon la branche :

$$r(s_i, a) = \begin{cases} C_i \times \dfrac{\sum_{k=1}^{\mathrm{Card}(T_i)} \mathrm{Eff}(t_{i,k})}{\mathrm{Card}(T_i)}, & \text{si branche liée à un type.} \\ \mathrm{Max}(\mathrm{Min}(\mathrm{score}_i + \alpha \cdot D_i - \alpha, -1), 1), & \text{si branche liée à la demande.} \\ \mathrm{Max}(\mathrm{Min}(\mathrm{score}_i + \alpha \cdot O_i - \alpha, -1), 1), & \text{si branche liée à l'offre.} \end{cases} \tag{4.11}$$

où :

— $\mathrm{score}_i = E_i + S_i + R_i$ représente l'avancement du système à l'itération i.

— T_i l'ensemble des tâches distribuées commencées à l'itération i.

— $\mathrm{Eff}(t)$ retourne l'efficacité de la tâche à savoir le rapport gain obtenu par rapport au gain maximal théorique.

— D_i et O_i étant respectivement les valeurs de batteries minimales observées de l'offre et de la demande à l'itération i

— α, une constante

Le second système récompense identiquement chaque branche :

$$r(s_i, a) = \frac{\mathrm{score}_i}{\alpha} + \mathrm{Min}(D_i, O_i) - \alpha \tag{4.12}$$

Bien que prometteuse, l'approche BDQ s'avère très récente et testée uniquement dans des contextes différents du notre. Par conséquent, nous avons souhaité la comparer à une approche multi-agents, seule autre alternative au vu du nombre d'actions à considérer.

4.4.3 Approches multi-agents

L'objectif de cette approche est la même que la précédente et l'intelligence doit conduire le même type d'actions. La différence réside dans la gestion de l'espace d'action qui est ici réalisée par division de l'agent en plusieurs sous-agents indépendants traitant chacun une partie de l'espace d'action. Nous proposons deux approches : une dite naïve où un sous-agent existe par sous-dimension et une, plus travaillée, où des dimensions ont été regroupées afin limiter le nombre d'agents.

Approche naïve

Dans cette approche, nous avons divisé le problème en sept agents traitants chacun une dimension comme illustré en figure 4.8. Cela reprend le principe de notre approche BDQ. Cependant, dans cette dernière, les branches partagent l'estimation des *states-values* ainsi que le réseau réalisant l'extraction des caractéristiques communes. Ici, seul l'état et la définition d'une récompense par équipe peuvent aider à la convergence. Les réseaux déployés sont identiques (hormis la couche finale qui dépend du nombre d'actions de la sous-dimension) et sont de types 3DQN avec quatre couches de 256 neurones communes et une couche de 128 neurones pour les branches *advantage-action* et *state-value*. La sélection de la meilleure action de chaque réseau indépendant permet de former l'action finale.

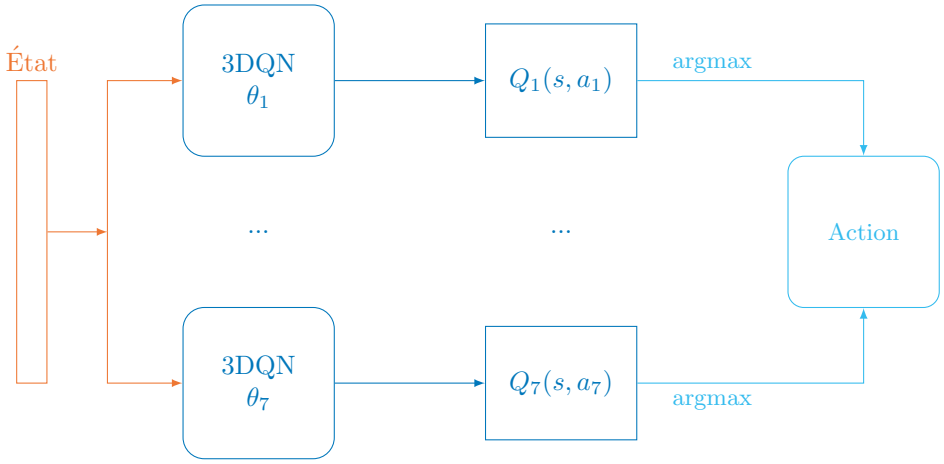

FIGURE 4.8 – Architecture multi-agents naïve : sept sous-agents sont déployés. Il ne possèdent en commun que le signal d'état transmis par l'environnement. La combinaison des actions de chaque sous-agent permet d'obtenir l'action finale sur le système.

Fort logiquement, nous avons obtenu de meilleures performances en ne faisant intervenir les sous-agents traitant les types que lorsqu'au moins une tâche du type était présente en offre de travailleur. En effet, les faire explorer/apprendre lorsque leur action ne peut avoir d'incidence nuit très fortement à la convergence du système. Cela illustre la nécessité de posséder une certaine expertise du domaine pour utiliser efficacement des méthodes d'apprentissages.

Nous avons utilisé deux schémas de récompense similaires à ceux utilisés pour l'approche BDQ en l'adaptant au contexte de réseaux indépendants. Le premier récompense différemment selon le sous-agent :

$$r(s_i, a) = \begin{cases} C_i \times \dfrac{\sum_{k=1}^{\mathrm{Card}(T_i)} \mathrm{Eff}(t_{i,k})}{\mathrm{Card}(T_i)}, & \text{si le sous-agent est lié à un type.} \\ \mathrm{Max}(\mathrm{Min}(\mathrm{score}_i + \alpha \cdot D_i - \alpha, -1), 1), & \text{si sous-agent traite la demande.} \\ \mathrm{Max}(\mathrm{Min}(\mathrm{score}_i + \alpha \cdot O_i - \alpha, -1), 1), & \text{si sous-agent s'occupe de l'offre.} \end{cases} \tag{4.13}$$

et le second récompense identiquement les sous-agents :

$$r(s_i, a) = \frac{\mathrm{score}_i}{\alpha} + \mathrm{Min}(D_i, O_i) - \alpha \tag{4.14}$$

Cette approche multi-agents à sept sous-agents nous ayant semblé optimiste en raison du problème de convergence que rencontre ce type de solution, nous avons développé une autre approche multi-agents plus raisonnable.

Approche travaillée

L'approche multi-agents pose des difficultés de convergence avec l'augmentation du nombre de sous-agents. Nous proposons donc une autre solution, plus raffinée, où nous avons regroupé des sous-dimensions pour, au final, ne diviser le problème qu'en trois sous-agents comme illustré en figure 4.9. Chaque sous-agent étant responsable de davantage de répercussions, il leur est plus facile de converger. La répartition est la suivante :

1. Le premier agent est responsable des tâches essentielles à l'avancement de la mission (exploration, recherche et sauvetage). Il est donc responsable de l'ensemble de la chaine de dépendances.
2. Le deuxième agent traite les types planification et fusion de données. Il est donc en charge de la partie pénalité.
3. Le dernier agent s'occupe de la partie énergétique.

Cette division du problème apporte de la cohérence aux sous-agents. Cependant, elle repose (dans une moindre mesure) la question de la taille de l'espace d'action. En effet, si les agents deux et trois n'ont que 49 actions à considérer, le premier agent est supposé traiter

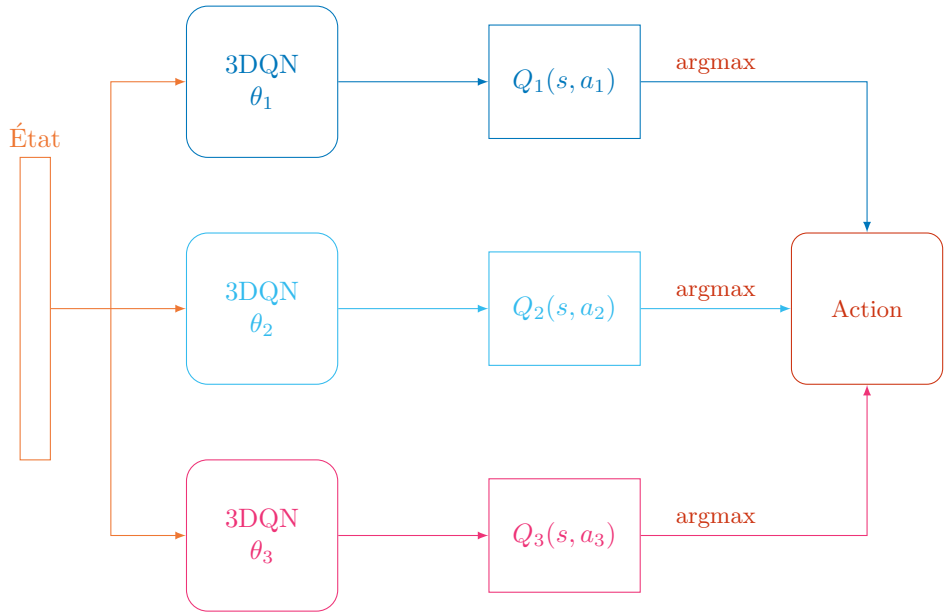

FIGURE 4.9 – Architecture multi-agents travaillée : trois sous-agents indépendants sont déployés. Ils traitent chacun une des parties du problème. La réunion de leurs actions forment l'action finale.

486 actions. Ce nombre d'actions, bien que relativement raisonnable, va considérablement compliquer la convergence du système. Pour remédier à cela, nous avons réduit le nombre d'actions possibles pour le premier agent à 80 en ne considérant que les combinaisons pertinentes (par exemple : il est inutile de permettre la combinaison 9 tâches d'exploration, 6 tâches de recherche et 9 tâches sauvetage car le système ne possède simplement pas les capacités pour toutes les exécuter simultanément).

Nous retrouvons nos deux schémas de récompense :
Celui qui récompense différemment selon le sous-agent :

$$r(s_i, a) = \begin{cases} C_i \times \dfrac{\sum_{k=1}^{\mathrm{Card}(T_i)} \mathrm{Eff}(t_{i,k})}{\mathrm{Card}(T_i)}, & \text{si le sous-agent est lié à un type.} \\ \mathrm{Max}(\mathrm{Min}(\mathrm{score}_i + \alpha \mathrm{Min}(D_i, O_i - \alpha, -1), 1), & \text{si sous-agent lié à la demande.} \end{cases} \tag{4.15}$$

et le second récompense identiquement les sous-agents :

$$r(s_i, a) = \frac{\text{score}_i}{\alpha} + \text{Min}(D_i, O_i) - \alpha \tag{4.16}$$

Avant de présenter les performances obtenues par les différentes solutions, nous allons décrire les conditions de simulation et les paramétrages utilisés.

4.5 Conditions d'expérimentation

Comme expliqué dans le chapitre 3, la complexité du problème rend tout déploiement réel impossible. Dans le cadre de la thèse, nous avons donc développé un simulateur afin d'évaluer la pertinence de ces approches et de choisir le cas échéant, la meilleure configuration.

4.5.1 Présentation du simulateur

Le changement de paradigme et la forte complexification du problème par rapport aux travaux précédents nous ont forcés à développer, pendant plusieurs mois, un nouveau simulateur open source qui sera bientôt mis à la disposition du public. Ce dernier, plus complexe, intègre les notions d'aires de mission, de jobs, d'effet d'une tâche sur l'environnement, de principe de mission et est capable de :

1. Simuler le déploiement d'un système multi-robots décentralisé dans un contexte de S&R.
2. Produire différentes missions en faisant varier des paramètres clés :
   (a) Caractéristiques de l'aire de mission
   (b) Paramètres (pré-requis des tâches) du jeu de tâches
   (c) Configuration du système multi-robots (taille, capacité de la batterie)
3. Offrir deux types de résolutions :
   (a) Classique au moyen de méthodes d'algorithmes d'enchères doubles.
   (b) Basé sur l'apprentissage par renforcement simple ou multi-agents à l'aide d'architectures 3DQN ou BDQ.

Le caractère fortement paramétrable de notre émulateur permet de simuler une grande variété de missions.

4.5.2 Paramétrages

Les natures séquentielles du problème et de l'apprentissage par renforcement occasionnent des temps de simulations conséquents. De plus, la présence d'aléas dans le processus de sélection du type, nous force à moyenner nos résultats allongeant considérablement le temps nécessaire à leur obtention (plusieurs semaines). Par conséquent, nous avons dû fixer arbitrairement un certain nombres de paramètres.

Paramétrage de l'environnement

Les paramètres fixés relatifs à l'environnement sont les suivants :

TABLE 4.8 – Paramétrage de l'environnement : liste des paramètres fixés de l'environnement

Paramètres	Valeurs
Taille du système multi-robots	20
Nombre d'itérations nécessaires à l'exécution d'une tâche	4
Nombre d'itérations entre chaque enchère	2
μ_1	0.01
μ_2	0.025
λ_P	1
λ_F	1

La raison principale de ce choix de paramètres réside dans la nécessité de trouver les bons paramètres d'apprentissage (ainsi que le système de récompense adéquat). Or, il n'existe pas d'autre solution que de procéder par essais successifs nécessitant un temps de simulation conséquent (trois mois).

Paramétrage de l'apprentissage

Nos solutions basées sur le *deep Q-learning* utilisent le mécanisme d'*expérience replay* lors de l'apprentissage. Dans le cas des approches multi-agents, chaque agent possède sa propre mémoire d'expériences. Nous obtenons les meilleures performances avec le paramétrage suivant :

TABLE 4.9 – Paramétrage de l'apprentissage : valeurs communes aux solutions

Paramètres	**Valeurs**
Learning rate : α	10^{-3}
Discount factor : γ	0.95
Mise à jour du *target network* : c	40
Taille de la mémoire d'expériences	256
Taille de l'échantillon mémoire utilisé	32

Quels que soient l'aire de la mission ou le jeu de tâches utilisés, les solutions basées sur l'apprentissage par renforcement sont préalablement entraînées sur 40 missions, aucun apprentissage n'est effectué pendant les évaluations.

Aires de missions et jeux de tâches

Nous avons créé trois aires (se référer au Tableau 4.2 pour connaître la définition de chacune des caractéristiques) afin d'observer les effets des différents paramètres sur le comportement des solutions. Sauf mention contraire, l'aire n°1 est utilisée.

TABLE 4.10 – Paramétrage des aires de mission

Paramètres	**Valeurs**		
	Aire n°1	**Aire n°2**	**Aire n°3**
Danger	0.5	0.75	0.25
Densité ROI	0.5	0.25	0.75
Exploration	0	0	0
Recherche	0	0	0
Paliers de sauvetage	[0.25, 0.5, 0.75, 1]	[0.33, 0.66, 1]	[0.2, 04, 0.6, 0.8, 1]
Sauvetage	0	0	0
Planification de trajectoires	0.25	0.25	0.25
Fusion de données	0.25	0.25	0.25

Toutes les aires sont considérées comme inexplorées et possèdent par conséquent des valeurs initiales d'exploration, recherche et sauvetage de 0. De plus, elles commencent toutes avec un certain degré (25%) de planification et de fusion à effectuer afin de représenter le caractère inconnu des déploiements S&R et l'absence de planification préalable (complète) due à une information parcellaire.

Nous avons défini trois jeux de tâches allant de l'équilibré pour le jeu n°1 avec un ratio $\frac{\text{CPU}}{\text{RAM}} = 1$ au déséquilibré jeu n°3 où le ratio $\frac{1}{5}$ apparaît comme présenté dans le tableau 4.11. Concernant les jobs serveurs, nous avons fixé les ressources nécessaires à leur exécution à 0.1 pour le CPU et la RAM (mémoire). Concernant les déplacements : les jobs serveur pour les types "exploration" et "sauvetage" et le job travailleur pour le type "recherche" nécessitent un déplacement. Par conséquent, leur exécution est exclusive sur le même drone.

TABLE 4.11 – Jeux de tâches : pré-requis de chaque job selon le type et le jeu de la tâche.

Types	**Jeu n°1**		**Jeu n°2**		**Jeu n°3**	
	CPU	**RAM**	**CPU**	**RAM**	**CPU**	**RAM**
Exploration	0.3	0.3	0.2	0.4	0.1	0.5
Recherche	0.3	0.3	0.4	0.2	0.5	0.1
Sauvetage	0.3	0.3	0.3	0.3	0.3	0.3
Planification de trajectoires	0.3	0.3	0.35	0.25	0.4	0.2
Fusion de données	0.3	0.3	0.25	0.35	0.2	0.4

Nous allons maintenant présenter les résultats obtenus avec les paramétrages mentionnés ci-dessus.

4.6 Résultats

Les résultats présentés ont été moyennés 10000 fois afin d'atténuer l'aléa provenant de la génération des tâches. Les appellations "BDQ", "IDQN-T", "IDQN-N" et "Marché" font respectivement références aux solutions par branchements, multi-agents travaillée, multi-agents naïve et à l'approche classique dénuée d'apprentissage par renforcement. Pour rappel, l'objectif principal du système est de terminer la mission. La vitesse de complétion, c'est à dire le nombre d'itération nécessaire à la réalisation de la mission, n'est qu'un objectif secondaire. Les principales métriques utilisées sont les suivantes :

1. Taux de succès : traduit le nombre de missions terminées, l'objectif principal. Une solution efficace possèdera un taux élevé.
2. Vitesse de complétion : le nombre d'itérations nécessaires à la réalisation d'une mission **terminée**. Une solution rapide minimisera ce nombre d'itérations.

3. Charge moyenne CPU/mémoire : le niveau de charge CPU/mémoire moyen du système lors de missions **terminées**.
4. Tâches exécutées : le nombre de tâches exécutées (selon le type) par le système lors de missions **terminées**.
5. Niveau de batterie : le niveau de batterie final des drones à la fin de missions **terminées**.

4.6.1 Exécution locale vs distribuée

Avant de présenter les performances des différentes solutions, il nous semble important d'attester de la pertinence d'une approche distribuée par rapport à une solution locale.

Principes

Afin de pouvoir comparer ces deux paradigmes, nous avons créé une solution locale dont les principes sont les suivants :

1. Les règles de génération des tâches sont identiques.
2. Les drones essayeront de distribuer localement (sur eux-mêmes) leurs tâches en créant autant de jobs travailleurs que leurs ressources le permettent.
3. Contrairement aux approches distribuées, aucun job serveur n'est nécessaire allégeant le coût d'exécution d'une tâche.

De plus, nous avons dû augmenter la capacité initiale des batteries (d'un facteur allant 2 à 7 en fonction du jeu de tâches) afin de lui permettre de finir (parfois) la mission, et donc, de fournir des indices de performances.

Performances

Concernant la capacité de l'approche locale à réussir la mission, les résultats sont sans appel puisque, pour un niveau de batterie "standard" (la capacité de batterie utilisée comme référence pour le reste de ces travaux), aucune mission n'est réussie d'où la nécessité d'augmenter la capacité.

Pour ce qui est de la vitesse de résolution, les résultats obtenus avec le jeu n°1 et illustré sur le Figure 4.10 ne montrent pas de lacune de l'approche locale puisqu'en moyenne, elle s'avère plus rapide que l'approche multi-agents travaillée. Nous constatons néanmoins une forte disparité dans ses temps de résolution avec des missions 45% plus lentes la rendant particulièrement peu fiable. Concernant les approches distribuées, les solutions "IDQN-N" et "Marché" affichent les meilleurs scores de vitesse.

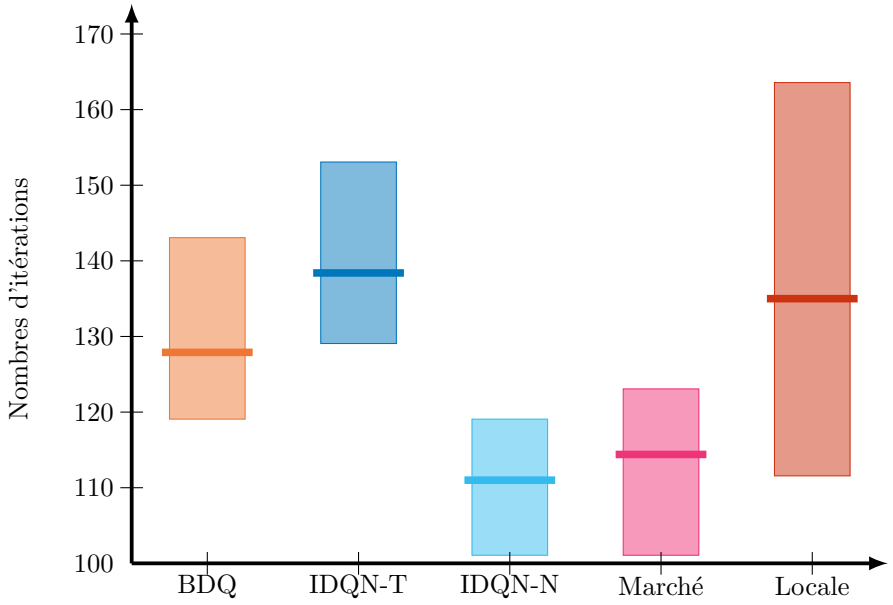

FIGURE 4.10 – Vitesses de résolution pour le jeu n°1 : chaque rectangle représente l'amplitude de vitesse entre la mission terminée le plus rapidement et celle terminée le plus lentement pour chaque approche. Les barres représentent la vitesse moyenne.

Cependant, lorsque nous regardons dans la figure 4.11 les scores de vitesse pour les jeux n°2 et n°3, les limites de l'approche locale apparaissent. En effet, contrairement au jeu n°1 qui permet à un drone d'exécuter localement une tâche avec trois travailleurs, les autres jeux, plus déséquilibrés limitent ce nombre à deux. Il en résulte une approche locale totalement inefficace. Concernant les approches distribuées, la solution "Marché" demeure plus rapide (la solution "IDQN-N" ne parvient pas à terminer la mission pour ces jeux de tâches), mais les écarts se réduisent avec les deux autres approches.

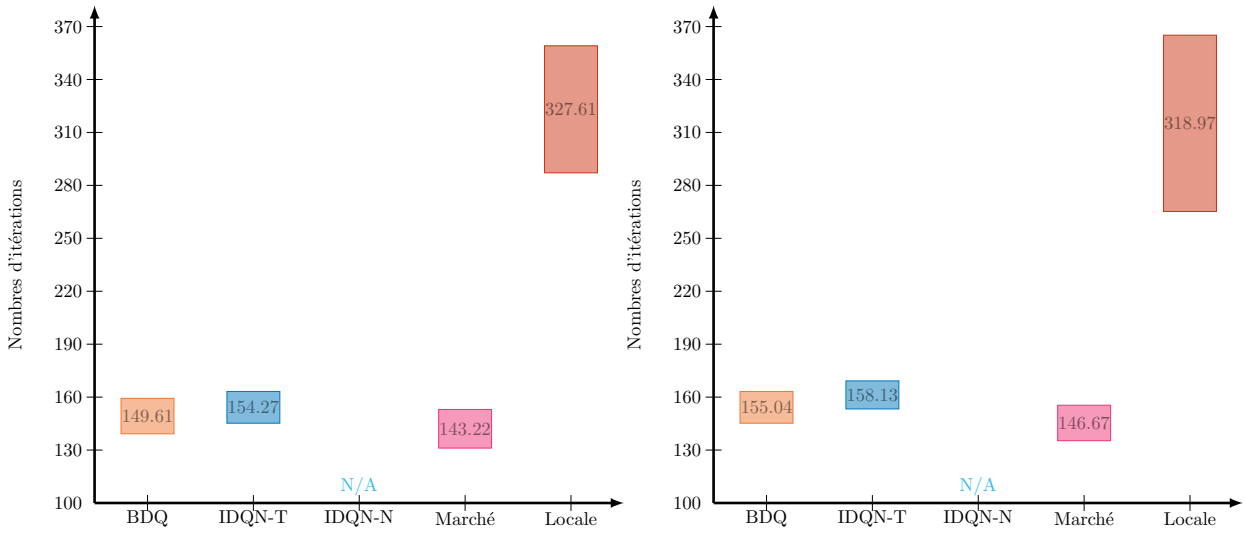

FIGURE 4.11 – Vitesses de résolution pour les jeux n°2 et n°3 : chaque rectangle représente l'amplitude de vitesse entre la mission terminée le plus rapidement et celle terminée le plus lentement pour chaque approche. Les valeurs présentent dans les rectangles indiquent les vitesses moyenne.

Les observations sont sans appel, l'approche locale est inapte à notre problème par conséquent, elle n'apparaitra pas comme élément de comparaison pour le reste de ces travaux.

☐ Les résultats observés jusqu'à présent attestent de la rapidité des approches : classique et multi-agents naïve, mais cette rapidité, implique-t-elle le succès de la mission ?

4.6.2 Taux de succès

Sauf mention contraire, les résultats présentés sont ceux recueillis avec une capacité de batterie de $6500W/mission$ pour un déploiement en aire n°1.

Taux de succès en fonction du jeu de tâches

La figure 4.12 affiche les taux de succès des différentes approches pour le jeu de tâche n°1. Les solutions "BDQ" et "IDQN-T" se révèlent très efficaces avec un taux d'échecs de 5%, nettement inférieur à celui de l'approche classique. Si la solution "IDQN-N" offre, elle

aussi, un taux d'échecs inférieur (néanmoins supérieur à 50 %), elle ne parvient pas à tirer partie de l'ensemble des mécanismes qui lui sont fournis.

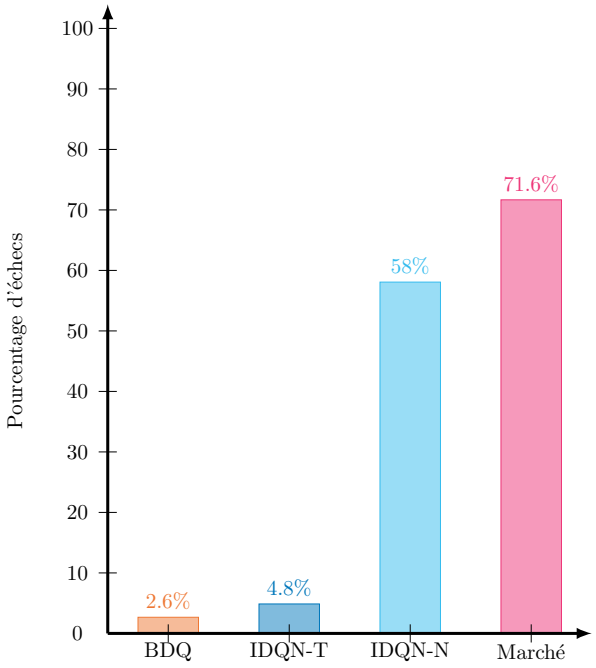

FIGURE 4.12 – Taux d'échecs de la mission pour le jeu n°1 pour une capacité initiale de batterie de 6500

☐ Que se passe-t-il lorsque la mission utilise un jeu de tâches plus déséquilibré ?

La figure 4.13 montre les taux de succès des différentes approches pour les jeux n°2 et n°3. Nous constatons directement que la disparité des besoins calculatoires introduite par ces jeux de tâches nuit à la qualité de la distribution. Par conséquent, les performances globales sont inférieures à celles obtenues pour le jeu de tâches n°1. La solution "BDQ" obtient les meilleurs performances pour le jeu n°2, mais souffre fortement de l'utilisation du troisième jeu avec un taux d'échecs trois fois supérieur à celui obtenu pour le jeu n°2. Nous ne retrouvons pas ce phénomène pour l'approche "IDQN-T" qui s'adapte aux changements de jeux et parvient à offrir les meilleurs performances pour le jeu n°3. La solution "Marché", bien qu'offrant un faible taux de succès, maintient ses scores en ne subissant qu'une faible augmentation du nombre d'échecs. Les choses se compliquent pour "IDQN-N" qui ne parvient pas à finir une seule mission attestant de l'absence de convergence, et donc, de l'échec de l'apprentissage.

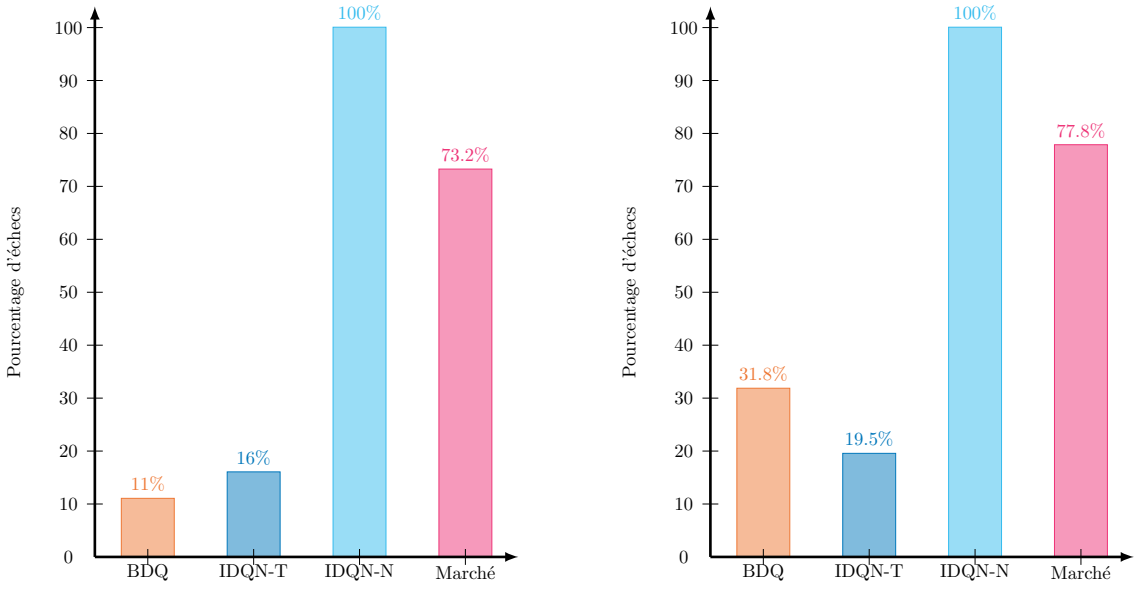

FIGURE 4.13 – Taux d'échecs de la mission pour le jeu n°2 (graphique de gauche) et n°3 (graphique de droite) pour une capacité initiale de batterie de 6500.

☐ Ces taux d'échecs, sont-ils confirmés lorsque la capacité initiale de la batterie varie ?

Taux de succès en fonction de la capacité de la batterie

La figure 4.14 montre le taux de succès des approches "BDQ" et "Marché" en fonction de la capacité de batterie initiale. L'utilisation de l'approche par renforcement permet d'obtenir un système parfaitement fiable pour un niveau de batterie supérieur à 6500. De plus, bien que présentant un taux d'échecs de 50 % pour des niveaux de batterie inférieurs, les performances observées demeurent similaires ou supérieures à celles de l'approche classique dotée de 500 de capacité de batterie supplémentaire.

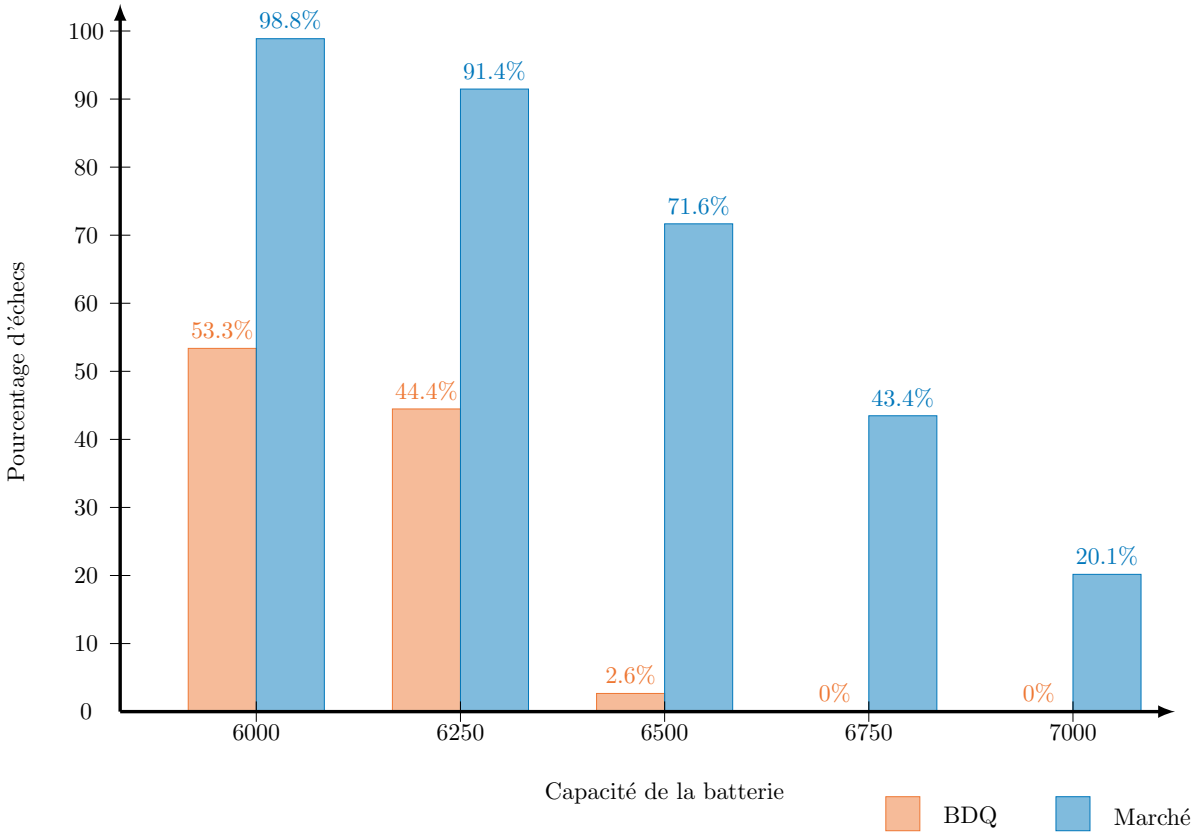

FIGURE 4.14 – Comparaison du taux d'échecs de la mission en fonction de la capacité initiale de la batterie pour les approches BDQ et marché avec le jeu de tâche n°1

Au vu des scores obtenus, l'intérêt de l'apprentissage par renforcement est manifeste.

☐ Ces performances résultent-t-elles, comme dans le chapitre précédent, d'une maximisation de l'utilisation des ressources calculatoires ?

4.6.3 Utilisation des ressources du système

Charge calculatoire du MRS

La figure 4.15 montre les charges moyennes des drones (processeur et mémoire) pour chaque approche au cours d'une mission effectuée avec le jeu de tâches n°1. L'approche "Marché" parvient à utiliser la quasi totalité des ressources du système attestant d'une distribution abondante à défaut d'être efficace. Des trois solutions utilisant l'apprentissage par renforcement, "IDQN-T" s'avère être la plus mesurée.

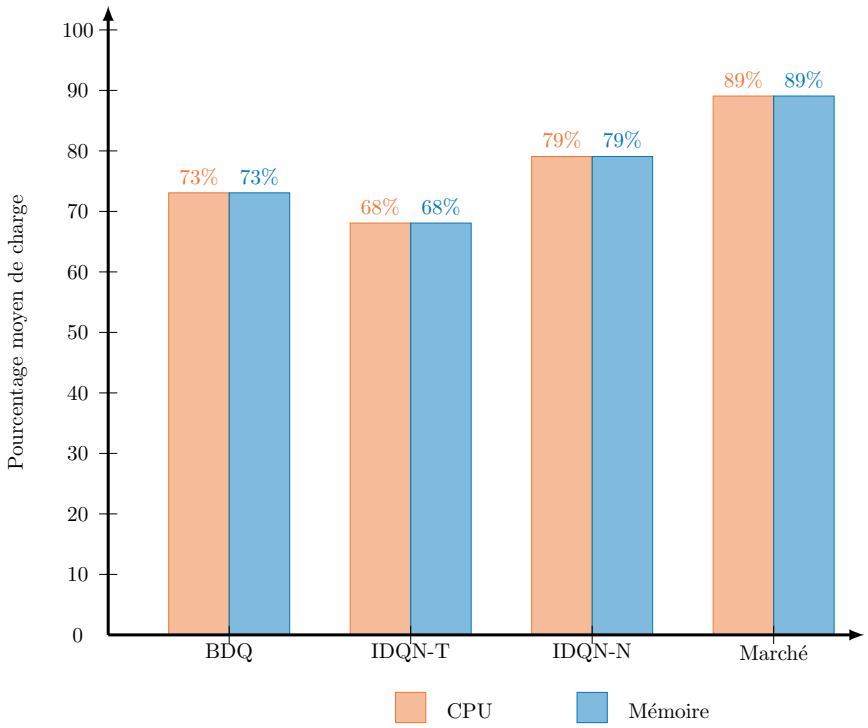

FIGURE 4.15 – Niveaux moyens de CPU (orange) et de mémoire (bleue) pour chaque approche lors d'une mission utilisant le jeu de tâches n°1.

La figure 4.16 affiche les niveaux moyens de charge CPU et mémoire pour chaque approche lors de missions utilisant les jeux 2 et 3. La solution "IDQN-N" échoue systématiquement pour ces jeux, par conséquent, ses niveaux sont absents des graphiques. Nous constatons, une fois de plus, l'effet des jeux sur les possibilités de distributions. En effet, de façon globale, les niveaux de charges diminuent sensiblement pour les jeux 2 et 3. Cela s'explique par la présence de phases d'exploration/recherche sollicitant davantage une res-

source que l'autre, générant un déséquilibre de charge et donc une sous-exploitation des ressources calculatoires. Nous remarquons aussi l'apparition d'un léger déséquilibre avec une ressource mémoire légèrement plus sollicitée. Cela indique que les tâches nécessitant plus de mémoire que de CPU sont plus souvent exécutées que leurs opposées.

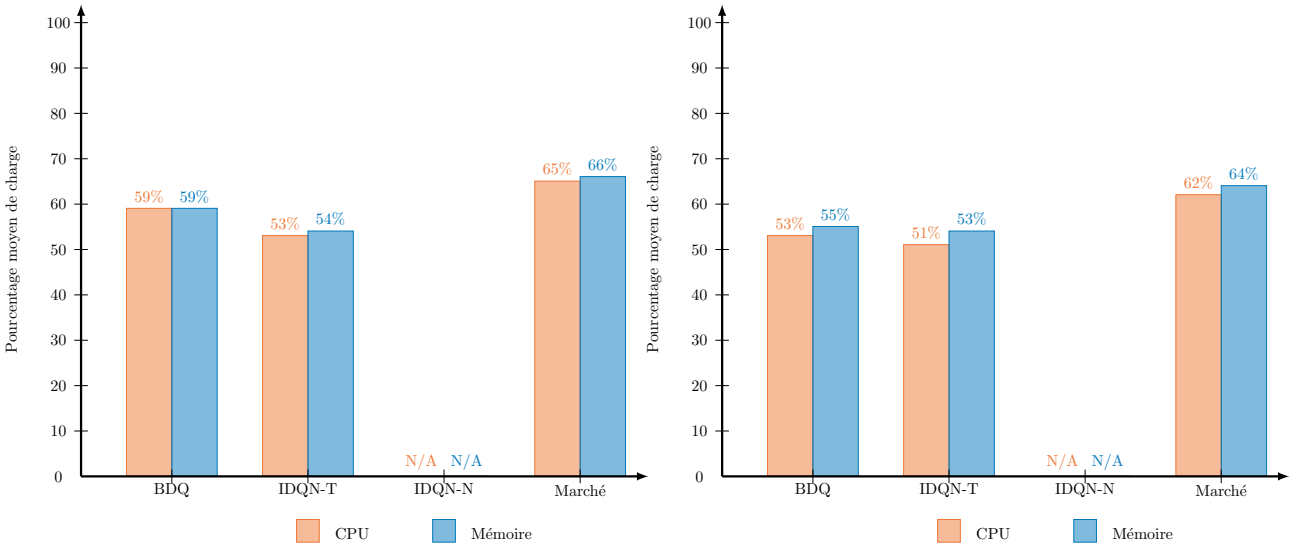

FIGURE 4.16 – Niveaux moyens de CPU (orange) et de mémoire (bleue) pour chaque approche lors d'une mission utilisant le jeu de tâches n°2 (graphique de gauche) et n°3 (graphique de droite). Échouant à terminer la mission, la solution "IDQN-N" n'offre aucun indice de performance.

S'il est manifeste que les solutions basées sur l'apprentissage par renforcement se refrènent d'allouer toutes leurs ressources, il est difficile de tirer davantage de conclusions de l'utilisation des ressources calculatoires.

- ☐ Si l'utilisation de davantage de ces ressources n'est pas synonyme de meilleurs performances, comment expliquer leurs taux de succès ?

Efficacité des tâches

L'efficacité d'une tâche (l'ampleur de son effet sur la mission) dépend de trois facteurs. Le premier est, bien entendu, la qualité de la distribution. Plus une tâche aura de travailleurs affectés à son exécution plus son effet sera démultiplié. Cependant, cette démultiplication ne sera efficace que si les besoins de planifications et de fusions sont faibles afin de limiter au maximum la pénalité encourue. Le dernier facteur concerne les tâches directement responsables de l'avancement de la mission qui doivent respecter la chaîne de dépendance, mais son incidence reste minime puisque les probabilités de sélection d'une tâche ne respectant pas cette chaîne s'avèrent extrêmement faibles.

La figure 4.17 expose le nombre de tâches essentielles effectuées par type et pour chaque solution lors d'une mission utilisant le jeu n°1. Les nombres totaux de tâches essentielles exécutées par les solutions s'avèrent relativement similaires avec un nombre moyen d'exécutions de 50 par type. Or, une efficacité parfaite nécessiterait 40 exécutions par types. Il en résulte donc une efficacité moyenne de 80% attestant d'une distribution efficace ainsi que d'une prise en considération de la pénalité.

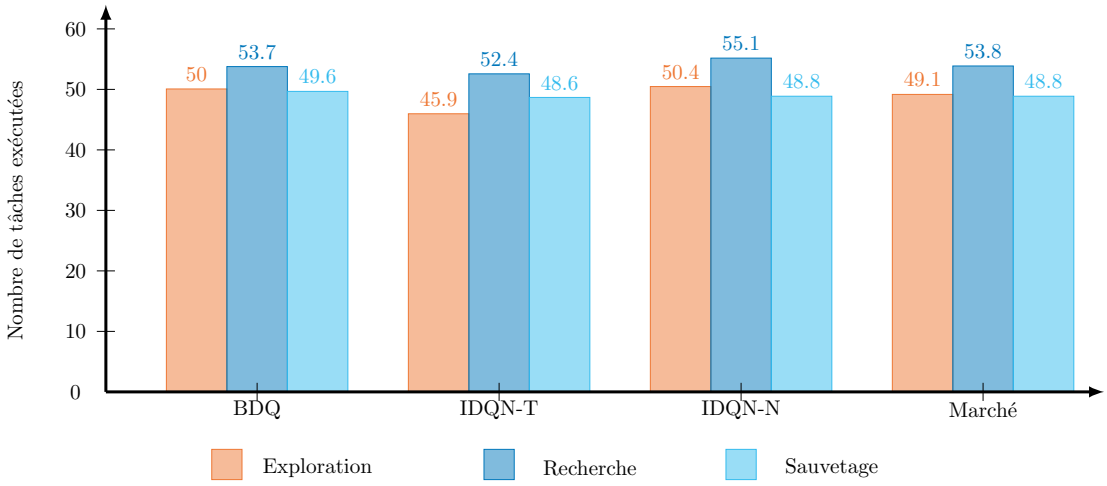

FIGURE 4.17 – Nombre de tâches essentielles exécutées par chaque solution pour une mission utilisant le jeu n°1 : la partie orange représente le type exploration, le type recherche est en bleu et le cyan décrit le type sauvetage

La figure 4.18 reflète l'efficacité des tâches essentielles pour les jeux n°2 et n°3. Globalement, nous constatons une efficacité moyenne stable pour toutes les solutions.

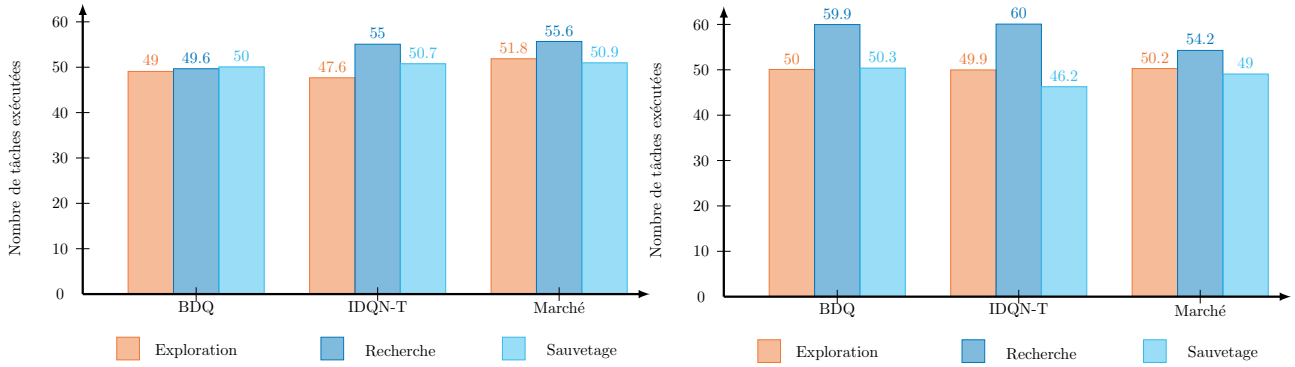

FIGURE 4.18 – Nombre de tâches essentielles exécutées par chaque solution pour les jeux de tâches n°2 (graphique de gauche) et n°3 (graphique de droite) : la partie orange représente le type exploration, le type recherche est en bleu et le cyan décrit le type sauvetage

☐ Les approches parviennent à s'adapter. Quelle est la situation des tâches non-essentielles ?

La figure 4.19 montre le nombre de tâches non-essentielles par type effectuées par chaque solution lors de missions utilisant les jeux n°1. La solution "Marché" exécute plus de tâches non essentielles ce qui corroborerait l'hypothèse d'une utilisation excessive des ressources. Cependant, comme observé précédemment, cela n'a pas d'effet sur l'efficacité des tâches nécessaires à la progression. De plus, ces types n'entrainent pas de déplacement, et donc, de forte consommation énergétique.

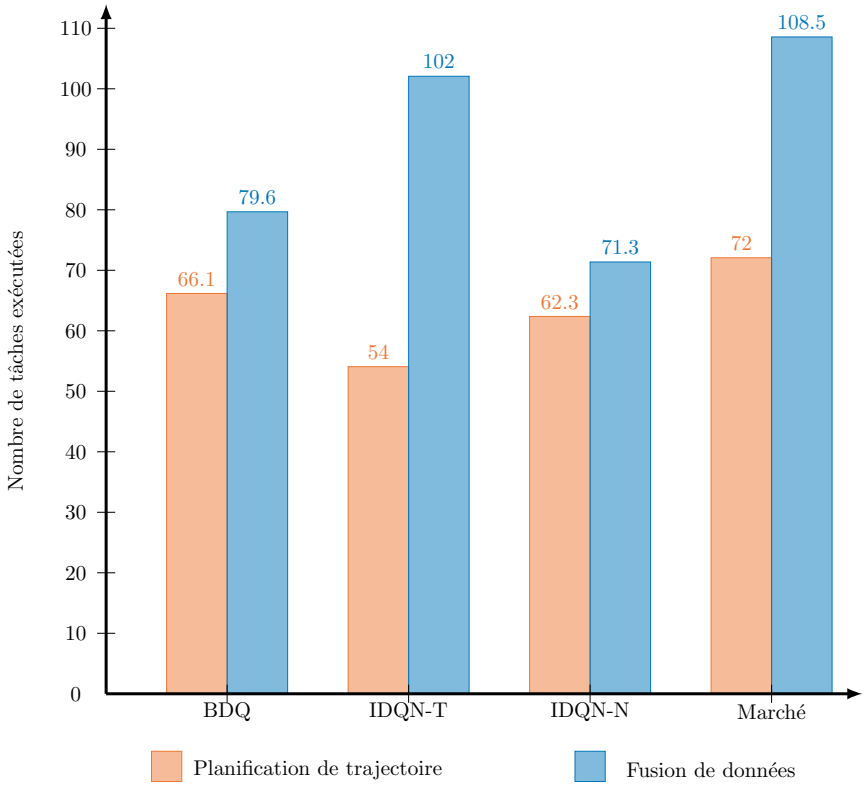

FIGURE 4.19 – Nombre de tâches non essentielles exécutées par chaque solution pour le jeu de tâches n°1 : la partie orange représente le type planification de trajectoire et la bleu décrit le type fusion de données

La figure 4.20 montre le nombre de tâches non-essentielles (planification de trajectoires et fusion de données) par type effectuées par chaque solution lors de missions utilisant les jeux n°2 et n°3. Nous observons une baisse globale du nombre d'exécutions de tâches de type fusion de données. Pourtant aucune baisse d'efficacité notable n'était observée pour ces jeux. Par conséquent, ces tâches étaient dispensables. Si les approches "BDQ" et "IDQN-N" semblent se comporter de façon identique, la solution "Marché" exécute environ 25% de tâches non-essentielles dont la conséquence reste à déterminer.

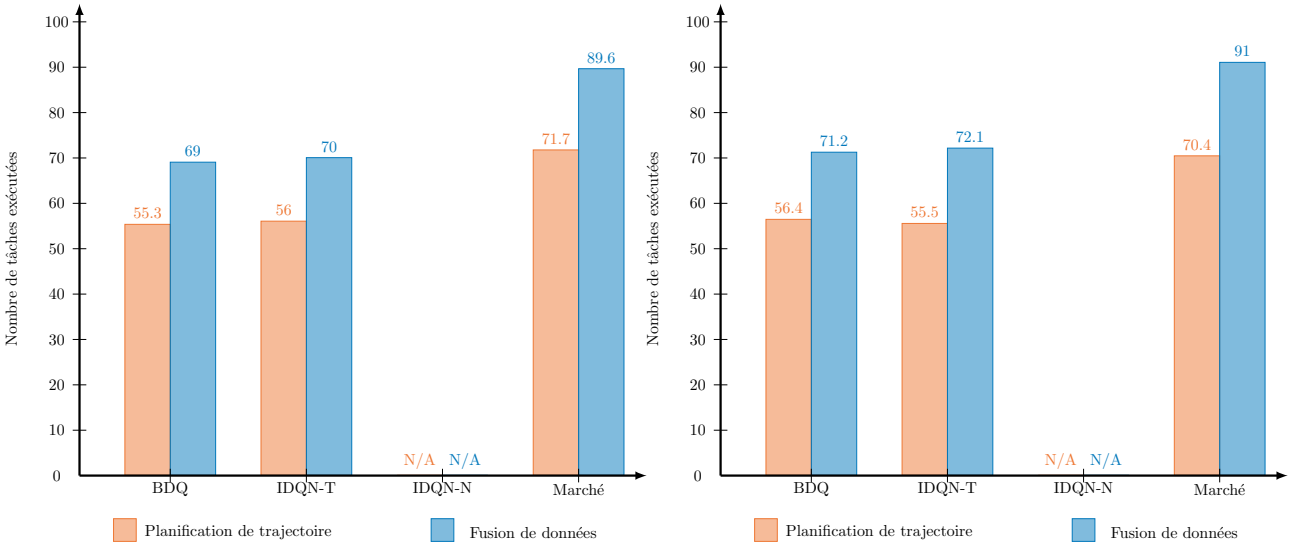

FIGURE 4.20 – Nombre de tâches non-essentielles exécutées par chaque solution pour les jeux de tâches n°2 (graphique de gauche) et n°3 (graphique de droite) : la partie orange représente le type planification de trajectoire et la bleue décrit le type fusion de données

Bien que présentant un faible taux de succès, la solution "Marché" exécute ses tâches efficacement.

☐ Dès lors, comment expliquer ses résultats ? S'agit-il d'une mauvaise répartition des tâches consommatrices ?

Qualité de la répartition des tâches énergivores

Nous pouvons analyser la qualité de la répartition des tâches énergivores en examinant les niveaux des batteries à la fin des missions terminées avec succès. La figure 4.21 expose les extremums et la moyenne des niveaux de batteries finaux dans le cas de missions terminées avec succès et ceux pour chaque approche. Les résultats sont sans équivoque. D'un côté, nous constatons que les méthodes efficaces (BDQ et IDQN-T) affichent peu d'amplitude entre le minimal et le maximal prouvant l'efficacité de leurs répartitions des tâches consommatrices. Cela atteste de la pertinence du critère énergétique pour disqualifier des demandes/offres. De même, la méthode classique offre d'excellents niveaux de batteries attestant de la pertinence de cette approche. Cela prouve, une fois de plus, que les piètres taux de succès obtenus par cette approche sont dus à sa nature statique la rendant incapable de maximiser l'efficacité en se refrénant d'allouer. De l'autre côté, l'approche IDQN-N affiche des niveaux de batteries fortement disparates résultant d'une

très mauvaise répartition des tâches énergivores. Cette surexploitation de certains drones conduit à l'échec de la mission et ce malgré une efficacité des tâches convaincantes.

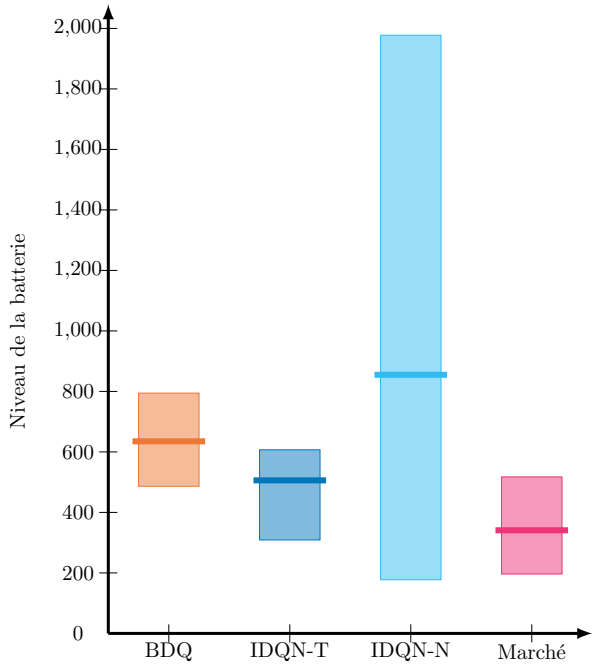

FIGURE 4.21 – Niveaux de batterie à l'issue de la mission pour le jeu de tâches n°1 : les rectangles représentent l'amplitude entre les niveaux de batteries finaux du drone possédant la batterie la plus déchargée et celui qui possède la plus chargée pour les missions terminées avec succès et ceux pour chaque approche. La barre représente le niveau de batterie moyen du système.

Ces observations sont confirmées par la figure 4.22 qui montre les niveaux de batterie finaux pour les jeux n°2 et n°3. L'ensemble des solutions ("IDQN-N" n'étant pas concernée) affichent de très faibles disparités de niveaux de batteries. Par conséquent, la qualité de la répartition des tâches énergivores de ces approches s'avèrent excellentes.

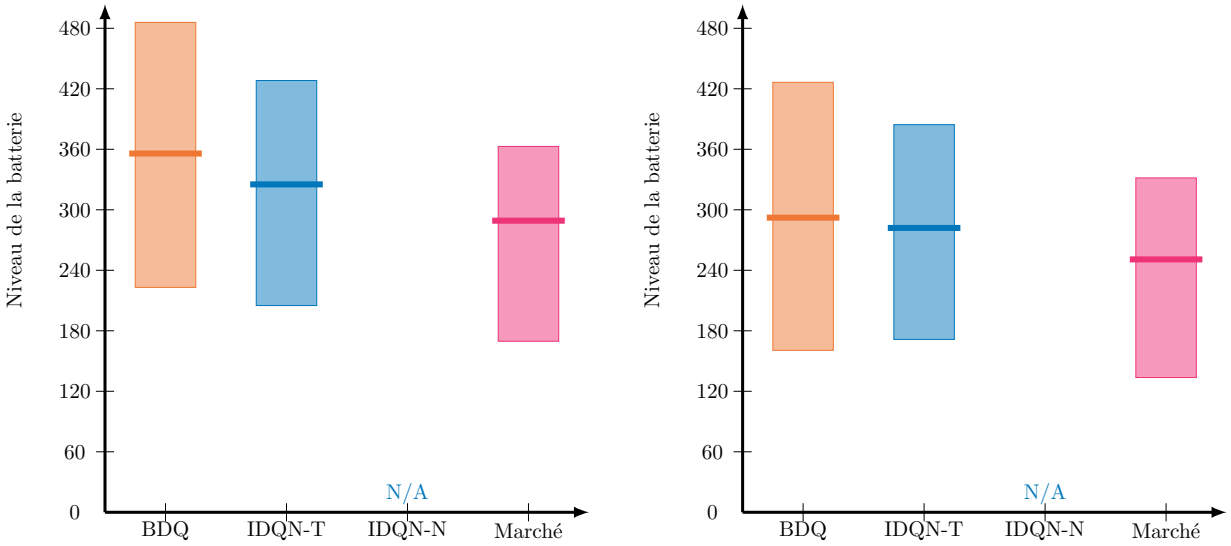

FIGURE 4.22 – Niveaux de batterie à l'issue de la mission pour le jeu de tâches n°2 (graphique de gauche) et n°3 (graphique de droite) : les rectangles représentent l'amplitude entre les niveaux de batteries finaux du drone possédant la batterie la plus déchargée et celui qui possède la plus chargée pour les missions terminées avec succès et ceux pour chaque approche. La barre représente le niveau de batterie moyen du système. Échouant à terminer la mission, la solution "IDQN-N" n'offre aucun indice de performance.

Notre approche "Marché" exécute efficacement ses tâches tout en répartissant équitablement le coût énergétique. Pourtant, ses taux d'échecs sont importants. Cela s'explique par une surexploitation (par rapport aux autres solutions) de ses ressources calculatoires lors de l'exécution de tâches non directement nécessaires à la complétion de la mission. Or, il s'agit des tâches les plus souvent exécutées et ceux, quelle que soit la solution utilisée. Nous constatons donc la difficulté de fixer un juste milieu. D'apparence, la solution "Marché" se comporte correctement, mais il est nécessaire d'utiliser une solution intelligente et adaptative pour saisir toute la complexité du problème. Il pourrait être tentant d'estimer que les piètres performances de cette approche sont dues au choix arbitraire de la capacité initiale de batterie. Mais nous avons observé avec la figure 4.14 que la solution "BDQ" offre de meilleurs résultats avec 500 de capacité de moins.

☐ Jusqu'à présent, nous n'avons pas abordé la question du système de récompense utilisé. Pourtant, nous en avons défini deux pour chaque approche. Quelles différences génèrent-ils et s'avèrent-ils équivalents ?

4.6.4 Systèmes de récompenses

Le système de récompenses conditionne fortement l'efficacité des méthodes d'apprentissage par renforcement. Jusqu'à présent, nous avons affiché les résultats du meilleur système de récompenses pour chaque jeu de tâches. La figure 4.23 expose les performances de chaque système en fonction du jeu de tâches pour les solutions "BDQ" et "IDQN-T". "BDQ" obtient un score légèrement supérieur avec le système 1 pour les jeux n°1 et n°2, mais un nettement moins bon score pour le jeu n°3. Pour "IDQN-T", le système n°2 est globalement préférable.

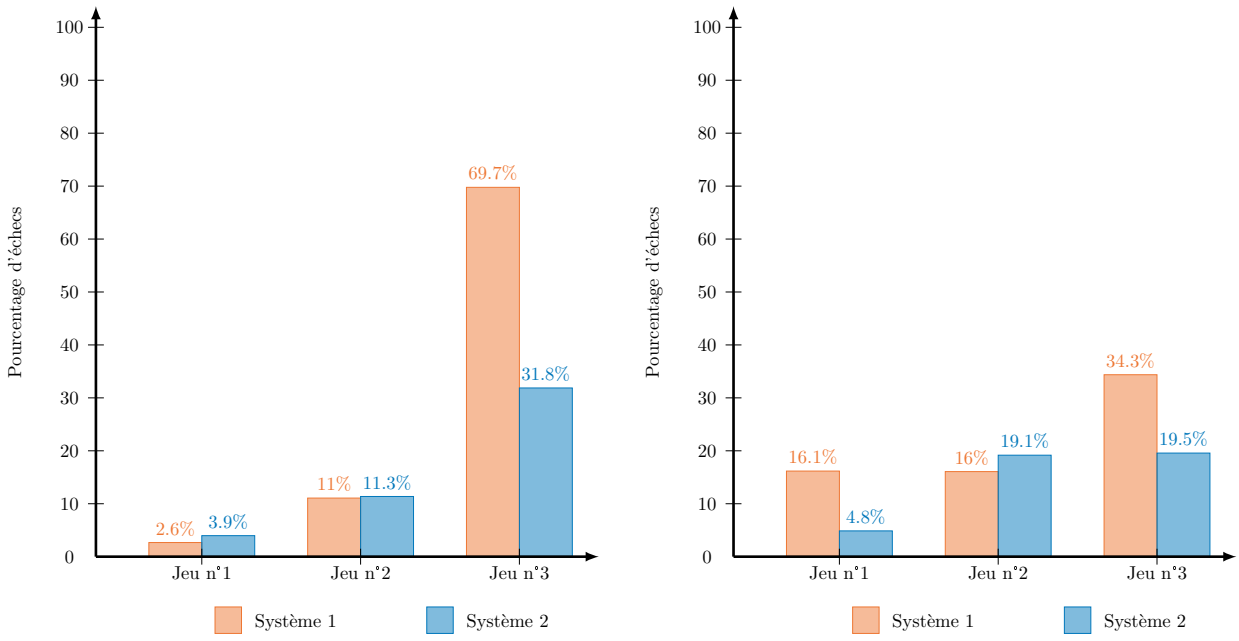

FIGURE 4.23 – Taux de succès en fonction du système de récompense utilisé pour l'approche "BDQ" (graphique de gauche) et "IDQN-T" (graphique de droite).

Nous pensons que la contre performance de "BDQ" pour le jeu n°3 s'explique par notre formule de calcul de cible TD qui confère le même poids à chaque branche. Or, il existe un fort déséquilibre entre les types de branches. Par conséquent, les branches relatives à l'énergie, par manque de poids, n'influent pas suffisamment sur la prise de décision. L'approche "IDQN-T" ne souffre pas de ce problème puisque les sous-agents sont indépendants.

Concernant l'approche "IDQN-N", seul l'apprentissage effectué sur le jeu de tâches n°1 pour le système 1 converge avec un score de 58%. Nous arrivons à obtenir quelques succès (2.6 %) avec le système 2. Cependant, le système ne converge pas, il s'agit de missions chanceuses !

4.7 Conclusion

Dans ce chapitre, nous avons exploré la gestion de l'allocation des ressources d'un cluster de robotique. Dans un souci de réalisme, nous avons ajouté un contexte de "recherche et sauvetage" afin de pouvoir modéliser les effets de la complétion de tâches sur l'environnement.

Gestion d'un cluster robotique

La gestion d'un cluster robotique s'avère complexe puisque toute utilisation des ressources se fait au détriment d'un autre membre du cluster. De plus, la quantité de ressources disponibles dépend de la charge libre de chacun de ses membres, et donc, varie inévitablement. Afin d'appréhender le caractère dynamique du problème, nous avons fortement complexifié notre modélisation en ajoutant :

1. Une chaîne de dépendances entre certaines tâches faisant varier la pertinence de leur exécution.
2. Une pénalité dynamique dépendant de besoins évoluant selon les actions du système multi-robots.
3. Un système de sélection, du type, basé sur des probabilités dynamiques liées à la compréhension de la situation par le drone.
4. Une contrainte énergétique forte conduisant éventuellement à l'échec de la mission.

La nature décentralisée du système, imposée par le contexte S& R, nécessite d'utiliser un processus décisionnel requérant peu de communications et ne générant pas de point de défaillance unique. Les méthodes basées sur le marché répondent à ce besoin. Cependant, contrairement aux autres travaux dans le domaine, notre problème nécessite de pouvoir mettre en concurrence simultanément les acheteurs et les vendeurs. Pour cela, nous avons recours au mécanisme d'enchères doubles qui s'apparente à un système boursier. La difficulté réside alors dans la définition de la méthode d'appareillements. Souhaitant pouvoir

discriminer selon deux critères (la charge calculatoire et la capacité énergétique), nous avons utilisé des offres multi-critères. Notre mécanisme de marché avait pour objectifs de permettre une distribution efficace des travailleurs ainsi qu'une répartition des tâches énergétiques. Il remplit parfaitement son office puisque les solutions proposées parviennent à atteindre les deux objectifs.

Bien que parvenant à exécuter ses tâches efficacement tout en répartissant celles fortement consommatrices d'énergie, notre solution classique ne parvient pas à offrir un taux de succès convaincant. Nous avons donc testé des approches utilisant l'apprentissage par renforcement pour affiner la conduite des enchères doubles, mais la complexité du problème génère un nombre important de combinaisons d'actions possibles.

Espaces d'action à fortes dimensions

Les méthodes d'apprentissage par renforcement capables de traiter des espaces d'action à fortes dimensions demeurent relativement peu explorées. Par conséquent, nous avons décidé d'étudier en parallèle les deux approches les plus prometteuses à savoir l'approche BDQ (branching dueling double deep Q-network) et l'approche multi-agents.

Les approches multi-agents souffrent de problèmes de convergence avec l'augmentation du nombre d'agents. Conscients de cette difficulté, nous avons testé deux architectures multi-agents. La première, plutôt naïve, est une transposition de notre approche BDQ où les agents remplacent les branches. Sans surprise, le nombre conséquent d'agents (sept) combiné à la finesse du problème a rendu la converge du système impossible dans la majorité des cas. Notre deuxième architecture diminuant le nombre d'agents en regroupant les actions par catégorie s'est montrée pertinente en offrant des résultats comparables à ceux obtenus avec l'approche BDQ. Pour atteindre ce niveau de performances, nous avons défini une méthode de récompense d'équipe afin de coordonner les agents. Néanmoins, cette approche multi-agents "regroupée" rencontre plusieurs limites :

1. En regroupant les actions par type, la problématique de la taille de l'espace d'action s'est reposée (dans une moindre mesure cependant), nous forçant à limiter son champ d'actions. Il existe donc un problème certain d'évolutivité.
2. De plus, ce regroupement fut possible par la nature du problème, nécessitant des connaissances préalables (que nous possédions grâce au comportement observé pour la solution BDQ). Cette solution s'avère donc problème dépendant.

3. Cette approche, par définition, multiplie le nombre d'agents, et donc, le nombre de réseaux de neurones et de mémoires d'expériences augmentant considérablement l'empreinte mémoire du système.

L'approche BDQ s'est révélée efficace en offrant de très bonnes performances. Seul le jeu de tâches n°2 a soulevé quelques difficultés. Il est possible que les formules utilisées pour calculer l'agrégation et la cible TD ne soient pas optimales pour ce problème. Cela proviendrait d'un déséquilibre de l'importance des branches. Quoi qu'il en soit, cette solution demeure stable et facilement déployable contrairement à l'approche multi-agents.

CONCLUSION ET PERSPECTIVES

Conclusion

La recherche croissante de systèmes multi-robots autonomes nécessite que ces derniers soient capables d'appréhender et d'analyser leur environnement en temps réel. Cette compréhension est rendue possible par l'augmentation de la taille des infrastructures de capteurs qui entraîne, inévitablement, un accroissement des traitements indispensables à la prise de décisions. Ces nouvelles tâches calculatoires deviennent un obstacle majeur au développement des systèmes multi-robots. Une solution consiste à distribuer ces tâches au sein du système ce qui nécessite de mettre en place un mécanisme d'allocations. Comme montré dans cette thèse, la distribution des tâches calculatoires est un problème complexe qui exige un processus décisionnel dynamique et adaptatif. Les approches basées sur l'apprentissage par renforcement offrent ce dynamisme et permettent aux systèmes multi-robots de d'appréhender l'incertitude.

Dans le premier chapitre, nous avons proposé une introduction à l'apprentissage par renforcement en présentant ses concepts clés ainsi que les approches de résolutions de type programmation dynamique et Monte Carlo. Nous avons également introduit les notions de neurones et de réseaux de neurones puisque ces derniers sont désormais couramment utilisés comme approximateurs de fonctions en apprentissage par renforcement.

Nous avons divisé le second chapitre en deux parties distinctes. Dans la première partie, nous avons présenté l'algorithme de *Q-learning* ainsi que sa récente amélioration, le *deep Q-learning*. Cette utilisation d'un réseau de neurones comme approximateur des *Q-values* permet de traiter des espaces d'état à fortes dimensions, mais entraîne l'instabilité de l'algorithme. Cette instabilité peut être corrigée en déployant un second réseau le *target network* combiné à une mémoire d'expériences. Cependant, un défi demeure, celui du traitement des espaces d'action à fortes dimensions. En effet, il est nécessaire de prendre en considération toutes les combinaisons d'actions possibles ce qui ralentit voire empêche toute convergence lorsque ce nombre devient trop important. À l'heure actuelle, deux

approches peu explorées ont été proposées pour contourner cette difficulté : l'approche par branches et l'approche multi-agents.

La deuxième partie de ce chapitre traitait de l'exécution et de la répartition de tâches calculatoires au sein d'un système multi-robot. Le problème de l'allocation des tâches au sein d'un système multi-robots (MRPTA) a déjà fait l'objet d'un grand nombre de travaux. Dans le cas de systèmes décentralisés, les approches basées sur le marché permettent une allocation efficace tout en limitant les communications et sans générer de point de défaillance unique. Cependant, nous avons constaté que ces traitements du MRTA posaient, comme principale contrainte, la localisation du robot par rapport à la tâche et non le coût calculatoire de cette dernière. Cette approche de l'allocation des tâches ne traite donc pas notre problématique. Néanmoins, la nécessité grandissante de traiter de lourdes tâches calculatoires à fait l'objet de travaux qui ont conduit à la création de deux paradigmes : le cloud robotique et le cluster robotique. Si le premier offre de nombreux avantages en permettant notamment de soulager le MRS de la charge de calcul, son utilisation s'avère impossible lorsqu'aucune connexion fiable n'est disponible ou lorsque le temps de latence constitue un facteur critique. Le deuxième paradigme mutualise les ressources calculatoires du système afin d'accélérer les tâches en utilisant les ressources libres présentes chez les robots peu chargés. Cette approche étant locale, elle s'affranchit des contraintes rencontrées par le cloud robotique. Cependant, les ressources mutualisées s'avèrent limitées et leur utilisation soulève de nombreux défis (répartition, respect des priorités, etc.).

Dans le troisième chapitre, nous avons défini une variante du MRTA, le MRpTA qui traite ce problème sous l'angle du coût calculatoire d'une tâche. Afin de valider notre hypothèse, nous avons exploré le déploiement d'un MRS décentralisé recevant périodiquement des tâches à exécuter. L'objectif du système était double puisqu'il devait compléter un maximum de tâches tout en favorisant les tâches prioritaires. Pour cela, le MRS était organisé en cluster robotique et pouvait transférer des tâches entre robots afin de répartir la charge au sein du système. Nous avons testé plusieurs solutions réparties en deux types. Les approches du premier type s'appuyaient sur un marché et des enchères séquentielles pour allouer les tâches transférées. Dans le second type, nous avons supprimé les enchères car la valorisation des offres constitue une limitation certaine en nécessitant un modèle de l'environnement. Pour cela, chaque robot était pourvu d'un DQN et devait apprendre par lui même quelle action entreprendre pour chaque tâche. De plus, aucune concerta-

tion préalable n'avait lieu lors d'un transfert forçant les robots à prédire les intentions des autres membres. Les résultats obtenus par simulation attestent de la pertinence de l'utilisation du Q-Learning comme processus décisionnel. Il offre un compromis intéressant entre quantité et respect des priorités. De plus, contrairement aux méthodes de résolutions classiques, sa nature dynamique lui permet de s'adapter aux éventuels déséquilibres.

Si le mécanisme de transfert s'avère intéressant, il ne s'agit pas de l'utilité première d'un cluster robotique qui est la parallélisation de tâches. Nous avons donc exploré, dans le quatrième chapitre, la répartition de la parallélisation de tâches au sein d'un cluster robotique. Pour cela, nous avons créé des missions de type recherche et sauvetage dans lesquelles un MRS décentralisé devait conduire différentes tâches pour réaliser la mission. Contrairement au troisième chapitre, les tâches pouvaient être parallélisées afin d'augmenter leurs effets, possédaient des dépendances et influaient directement sur l'environnement ce qui pouvait conduire à la paralysie du système. De plus, nous avons intégré une contrainte forte au système puisqu'il devait terminer la mission avant de tomber à court d'énergie.

La répartition de la parallélisation s'effectuant à l'aide d'une approche basée sur le marché et d'enchères doubles, le commissaire-priseur pouvait influer sur la conduite des tâches du système. Nous avons donc testé plusieurs approches de répartition par le commissaire-priseur : classique, BDQ et multi-agents. Bien que proposant un schéma d'allocations performant ainsi qu'une bonne répartition de la consommation, l'approche classique s'est montrée incapable d'appréhender les difficultés du problème et a montré de piètres résultats. L'approche par branches (BDQ), s'est avérée être la plus performante offrant globalement les meilleurs résultats et surpassant, de loin, l'approche classique. Les approches multi-agents ont produit des résultats diamétralement opposés. D'un côté l'approche naïve s'est révélée instable et incapable de converger proposant, dans de nombreux cas, des résultats nettement inférieurs à l'approche classique. De l'autre côté, l'approche par regroupement de dimensions présente d'excellentes performances parvenant même à surpasser dans certains cas l'approche par branches.

Dans ces travaux, nous avons étudié les apports de méthodes d'apprentissages pour la distribution de tâches au sein d'un cluster robotique. Nous avons opté pour l'algorithme de *Q-learning*, et plus particulièrement, son utilisation avec un réseau de neurones. La thématique des clusters robotiques et plus généralement celle des tâches calculatoires

exécutées au sein d'un MRS ont reçu peu d'attention. Nous avons donc dû définir une variante originale du MRTA afin de connecter le concept de cluster à ce problème. Dans ce cadre, il nous apparaît clair que des approches de types *deep Q-learning* peuvent servir de substitut aux processus décisionnels classiques ou bien de complément afin d'affiner la prise de décisions. Par ailleurs, nos expérimentations nous ont conduit aux problèmes des espaces d'action discrets à fortes dimensions pour lequel nous avons testé deux solutions. La première, l'approche multi-agents, se heurte très vite à l'explosion du nombre d'agents forçant à regrouper les dimensions ce qui nécessite une certaine compréhension du système ainsi que l'existence de liens entre les dimensions. La deuxième solution, l'approche par branches nous paraît nettement plus fiable, évolutive et adaptable à tout problème.

Perspectives

Cette section présente les perspectives les plus pertinentes ouvertes par nos travaux. La thématique traitée a fait l'objet de peu d'études alors que le besoin existe. Par conséquent, nous ne sommes qu'au début de la recherche et de nombreuses améliorations peuvent être apportées. Une barrière importante existe cependant concernant la validation des hypothèses et l'obtention des résultats. En effet, les systèmes multi-robots actuels ne sont pas suffisamment performants pour tester de tels scénarios. Il est nécessaire de recourir à de longues et complexes simulations.

La première perspective serait d'affiner notre modèle. En effet, nos simulations font appels à de nombreux concepts complexes que nous avons dûs simplifier. Il est donc possible d'affiner les simulations à l'aide de modélisation plus fidèles des composants électroniques, du comportement des tâches, de la gestion des communications, etc. Cela représente un défi en soi car la granularité de la discrétisation de la temporalité utilisée peut considérablement réduire la marge de manœuvre.

La seconde perspective consisterait à unifier notre problème du MRpTA avec la variante classique du MRTA afin que le problème prenne en considération la charge calculatoire et la position du robot. Il s'agit d'une étape fondamentale puisque les pré-requis positionnel et calculatoire sont quasi-systématiques en robotique. Cette unification nécessitera de combiner deux paradigmes aux contraintes et aux échelles de temps fondamentalement différentes. Cependant, le temps reste une notion commune aux deux aspects et pourra

servir de passerelle entre ces paradigmes.

La dernière perspective, probablement la plus importante mais aussi la plus difficilement réalisable, serait de tester nos hypothèses sur un système multi-robots physiques. Cependant, comme nous l'avons déjà mentionné, les MRS autonomes de 20 entités capables d'exécuter différentes tâches parallélisables n'existent tout simplement pas. Une étape intermédiaire pourrait être de simuler en *hardware in the loop* à l'aide d'un simulateur dynamique 3D tel que Gazebo [3].

Publications associées à la thèse

Cette thèse a donné lieu à trois publications/soumissions :

1. Le Chapitre n°3 a fait l'objet d'une publication à IEEE IRC.
2. L'article d'IRC a été retenu pour une version étendue dans IJTransAI. Cet article présente une étude approfondie qui s'appuie sur de nouvelles simulations.
3. Le Chapitre 4 est résumé dans un article soumis à IEEE Transactions on Automation Science and Engineering.

3. Gazebo est un simulateur 3D, cinématique et dynamique permettant de simuler des robots articulés dans des environnements complexes en trois dimensions [19].

BIBLIOGRAPHIE

[1] Richard Bellman, « A Markovian Decision Process », en, in : *Indiana Univ. Math. J.* 6.*4* (1957), p. 679-684, issn : 0022-2518, doi : 10.1512/iumj.1957.6.56038.

[2] Dmitry Berenson, Pieter Abbeel et Ken Goldberg, « A robot path planning framework that learns from experience », en, in : *2012 IEEE International Conference on Robotics and Automation*, St Paul, MN, USA : IEEE, mai 2012, p. 3671-3678, isbn : 978-1-4673-1405-3 978-1-4673-1403-9 978-1-4673-1578-4 978-1-4673-1404-6, doi : 10.1109/ICRA.2012.6224742.

[3] Leonardo Camargo-Forero, Pablo Royo et Xavier Prats, « Towards high performance robotic computing », en, in : *Robotics and Autonomous Systems* 107 (sept. 2018), p. 167-181, issn : 09218890, doi : 10.1016/j.robot.2018.05.011.

[4] CLEARPATH, *clearpath_heron_usermanual.pdf*, 2020.

[5] M. B. Dias et al., « Market-Based Multirobot Coordination : A Survey and Analysis », in : *Proceedings of the IEEE* 94.*7* (juill. 2006), p. 1257-1270, issn : 0018-9219, doi : 10.1109/JPROC.2006.876939.

[6] Brian P. Gerkey et Maja J. Matarić, « A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems », en, in : *The International Journal of Robotics Research* 23.*9* (sept. 2004), p. 939-954, issn : 0278-3649, 1741-3176, doi : 10.1177/0278364904045564.

[7] C. Goldfeder et al., « The Columbia grasp database », en, in : *2009 IEEE International Conference on Robotics and Automation*, Kobe : IEEE, mai 2009, p. 1710-1716, isbn : 978-1-4244-2788-8, doi : 10.1109/ROBOT.2009.5152709.

[8] Bruno Duarte Gouveia et al., « Computation Sharing in Distributed Robotic Systems : A Case Study on SLAM », en, in : *IEEE Trans. Automat. Sci. Eng.* 12.*2* (avr. 2015), p. 410-422, issn : 1545-5955, 1558-3783, doi : 10.1109/TASE.2014.2357216.

[9] Robert Grandl et al., « Multi-resource packing for cluster schedulers », en, in : *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*, Chicago, Illinois, USA : ACM Press, 2014, p. 455-466, isbn : 978-1-4503-2836-4, doi : 10.1145/2619239.2626334.

[10] Hado van Hasselt, Arthur Guez et David Silver, « Deep Reinforcement Learning with Double Q-learning », en, in : *arXiv :1509.06461 [cs]* (déc. 2015), arXiv : 1509.06461.

[11] Hado V Hasselt, « Double Q-learning », en, in : (2010), p. 9.

[12] George Iosifidis et al., « A Double-Auction Mechanism for Mobile Data-Offloading Markets », en, in : *IEEE/ACM Trans. Networking* 23.5 (oct. 2015), p. 1634-1647, issn : 1063-6692, 1558-2566, doi : 10.1109/TNET.2014.2345875.

[13] Tommi Jaakkola, Michael I Jordan et Satinder P Singh, « Convergence of Stochastic Iterative Dynamic Programming Algorithms », en, in : (1994), p. 8.

[14] X. Jia et M. Q.- Meng, « A survey and analysis of task allocation algorithms in multi-robot systems », in : *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, déc. 2013, p. 2280-2285, doi : 10.1109/ROBIO.2013.6739809.

[15] Junqi Jin et al., « Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising », en, in : *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (oct. 2018), arXiv : 1802.09756, p. 2193-2201, doi : 10.1145/3269206.3272021.

[16] U. Kant et D. Grosu, « Double auction protocols for resource allocation in grids », en, in : *International Conference on Information Technology : Coding and Computing (ITCC'05) - Volume II*, Las Vegas, NV, USA : IEEE, 2005, 366-371 Vol. 1, isbn : 978-0-7695-2315-6, doi : 10.1109/ITCC.2005.135.

[17] Ben Kehoe et al., « A Survey of Research on Cloud Robotics and Automation », en, in : *IEEE Trans. Automat. Sci. Eng.* 12.2 (avr. 2015), p. 398-409, issn : 1545-5955, 1558-3783, doi : 10.1109/TASE.2014.2376492.

[18] Ben Kehoe et al., « Cloud-based robot grasping with the google object recognition engine », en, in : *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany : IEEE, mai 2013, p. 4263-4270, isbn : 978-1-4673-5643-5 978-1-4673-5641-1, doi : 10.1109/ICRA.2013.6631180.

[19] N. Koenig et A. Howard, « Design and use paradigms for gazebo, an open-source multi-robot simulator », en, in : *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, t. 3, Sendai, Japan : IEEE, 2004, p. 2149-2154, ISBN : 978-0-7803-8463-7, DOI : 10.1109/IROS.2004.1389727.

[20] Sven Koenig, « The Power of Sequential Single-Item Auctions for Agent Coordination », en, in : (2006), p. 5.

[21] Sergey Levine et al., « End-to-End Training of Deep Visuomotor Policies », en, in : *arXiv :1504.00702 [cs]* (avr. 2016), arXiv : 1504.00702.

[22] Timothy P. Lillicrap et al., « Continuous control with deep reinforcement learning », en, in : *arXiv :1509.02971 [cs, stat]* (juill. 2019), arXiv : 1509.02971.

[23] Long-Ji Lin, « Reinforcement Learning for Robots Using Neural Networks », en, thèse de doct., jan. 1993.

[24] A Rupam Mahmood, « Weighted importance sampling for off-policy learning with linear function approximation », en, in : (2014), p. 9.

[25] Hongzi Mao et al., « Resource Management with Deep Reinforcement Learning », en, in : *Proceedings of the 15th ACM Workshop on Hot Topics in Networks - HotNets '16*, Atlanta, GA, USA : ACM Press, 2016, p. 50-56, ISBN : 978-1-4503-4661-0, DOI : 10.1145/3005745.3005750.

[26] Ali Marjovi, Sarvenaz Choobdar et Lino Marques, « Robotic clusters : Multi-robot systems as computer clusters », en, in : *Robotics and Autonomous Systems* 60.*9* (sept. 2012), p. 1191-1204, ISSN : 09218890, DOI : 10.1016/j.robot.2012.05.007.

[27] Matarić, « Multi-Robot Task Allocation in Uncertain Environments », in : 2003.

[28] Laetitia Matignon, Guillaume J. Laurent et Nadine Le Fort-Piat, « Independent reinforcement learners in cooperative Markov games : a survey regarding coordination problems », en, in : *The Knowledge Engineering Review* 27.*1* (fév. 2012), p. 1-31, ISSN : 0269-8889, 1469-8005, DOI : 10.1017/S0269888912000057.

[29] R.Preston McAfee, « A dominant strategy double auction », en, in : *Journal of Economic Theory* 56.*2* (avr. 1992), p. 434-450, ISSN : 00220531, DOI : 10.1016/0022-0531(92)90091-U.

[30] Warren McCulloch et Pitts Walter, « A logical calculus of the ideas immanent in nervous activity », en, in : (1943), p. 19.

[31] Luis Mejias et al., « Embedded Computation Architectures for Autonomy in Unmanned Aircraft Systems (UAS) », en, in : *Sensors* 21.*4* (fév. 2021), p. 1115, ISSN : 1424-8220, DOI : 10.3390/s21041115.

[32] Volodymyr Mnih et al., « Human-level control through deep reinforcement learning », en, in : *Nature* 518.*7540* (fév. 2015), p. 529-533, ISSN : 0028-0836, 1476-4687, DOI : 10.1038/nature14236.

[33] Volodymyr Mnih et al., « Playing Atari with Deep Reinforcement Learning », en, in : (déc. 2013).

[34] K. P. Naveen et Rajesh Sundaresan, « A double-auction mechanism for mobile data-offloading markets with strategic agents », en, in : *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, Shanghai : IEEE, mai 2018, p. 1-8, ISBN : 978-3-903176-00-3, DOI : 10.23919/WIOPT.2018.8362857.

[35] OpenAI et al., « Dota 2 with Large Scale Deep Reinforcement Learning », en, in : *arXiv :1912.06680 [cs, stat]* (déc. 2019), arXiv : 1912.06680.

[36] L. Riazuelo, Javier Civera et J.M.M. Montiel, « C2TAM : A Cloud framework for cooperative tracking and mapping », en, in : *Robotics and Autonomous Systems* 62.*4* (avr. 2014), p. 401-413, ISSN : 09218890, DOI : 10.1016/j.robot.2013.11.007.

[37] David E Rumelhart, Geoffrey E Hintont et Ronald J Williams, « Learning representations by back-propagating errors », en, in : (1986), p. 4.

[38] Javier Salmeron-Garcia et al., « A Tradeoff Analysis of a Cloud-Based Robot Navigation Assistant Using Stereo Image Processing », en, in : *IEEE Trans. Automat. Sci. Eng.* 12.*2* (avr. 2015), p. 444-454, ISSN : 1545-5955, 1558-3783, DOI : 10.1109/TASE.2015.2403593.

[39] Parnia Samimi, Youness Teimouri et Muriati Mukhtar, « A combinatorial double auction resource allocation model in cloud computing », en, in : *Information Sciences* 357 (août 2016), p. 201-216, ISSN : 00200255, DOI : 10.1016/j.ins.2014.02.008.

[40] Tuomas Sandholm, « Algorithm for optimal winner determination in combinatorial auctions », en, in : *Artificial Intelligence* (2002), p. 54.

[41] Tom Schaul et al., « Prioritized Experience Replay », en, in : *arXiv :1511.05952 [cs]* (fév. 2016), arXiv : 1511.05952.

[42] Julian Schrittwieser et al., « Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model », en, in : *arXiv :1911.08265 [cs, stat]* (fév. 2020), arXiv : 1911.08265.

[43] David Silver et al., « A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play », en, in : *Science* 362.*6419* (déc. 2018), p. 1140-1144, ISSN : 0036-8075, 1095-9203, DOI : 10.1126/science.aar6404.

[44] David Silver et al., « Mastering the game of Go without human knowledge », en, in : *Nature* 550.*7676* (oct. 2017), p. 354-359, ISSN : 0028-0836, 1476-4687, DOI : 10.1038/nature24270.

[45] Richard S. Sutton, « Learning to predict by the methods of temporal differences », en, in : *Mach Learn* 3.*1* (août 1988), p. 9-44, ISSN : 0885-6125, 1573-0565, DOI : 10.1007/BF00115009.

[46] Ardi Tampuu et al., « Multiagent Cooperation and Competition with Deep Reinforcement Learning », en, in : *arXiv :1511.08779 [cs, q-bio]* (nov. 2015), arXiv : 1511.08779.

[47] Zhu Tan et John R. Gurd, « Market-based grid resource allocation using a stable continuous double auction », en, in : *2007 8th IEEE/ACM International Conference on Grid Computing*, Austin, TX, USA : IEEE, sept. 2007, p. 283-290, ISBN : 978-1-4244-1559-5 978-1-4244-1560-1, DOI : 10.1109/GRID.2007.4354144.

[48] Arash Tavakoli, Fabio Pardo et Petar Kormushev, « Action Branching Architectures for Deep Reinforcement Learning », en, in : *arXiv :1711.08946 [cs]* (jan. 2019), arXiv : 1711.08946.

[49] Sebastian Thrun et Anton Schwartz, « Issues in Using Function Approximation for Reinforcement Learning », en, in : (1993), p. 9.

[50] John N Tsitsiklis, « Asynchronous Stochastic Approximation and Q-Learning », en, in : (1994), p. 18.

[51] Louis Turnbull et Biswanath Samanta, « Cloud robotics : Formation control of a multi robot system utilizing cloud infrastructure », en, in : *2013 Proceedings of IEEE Southeastcon*, Jacksonville, FL, USA : IEEE, avr. 2013, p. 1-4, ISBN : 978-1-4799-0053-4 978-1-4799-0052-7 978-1-4799-0051-0, DOI : 10.1109/SECON.2013.6567422.

[52] Jiafu Wan et al., « Cloud Robotics : Current Status and Open Issues », en, in : *IEEE Access* (2016), p. 1-1, ISSN : 2169-3536, DOI : 10.1109/ACCESS.2016.2574979.

[53] Z. Wang et al., « A task allocation algorithm based on market mechanism for multiple robot systems », in : *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, juin 2016, p. 150-155, DOI : 10.1109/RCAR.2016.7784017.

[54] Ziyu Wang et al., « Dueling Network Architectures for Deep Reinforcement Learning », en, in : *arXiv :1511.06581 [cs]* (avr. 2016), arXiv : 1511.06581.

[55] Watkins, « Learning from Delayed Rewards », thèse de doct., 1989.

[56] Watkins et Dayan., « Machine Learning », in : (1992).

[57] Elmar Wolfstetter, « AUCTIONS : AN INTRODUCTION », en, in : *J Economic Surveys* 10.*4* (déc. 1996), p. 367-420, ISSN : 0950-0804, 1467-6419, DOI : 10.1111/j.1467-6419.1996.tb00018.x.

[58] Chanmin Yoon et al., « Accurate power modeling of modern mobile application processors », en, in : *Journal of Systems Architecture* 81 (nov. 2017), p. 17-31, ISSN : 13837621, DOI : 10.1016/j.sysarc.2017.10.001.

[59] Jingchao Zhao et al., « A Search-and-Rescue Robot System for Remotely Sensing the Underground Coal Mine Environment », en, in : *Sensors* 17.*10* (oct. 2017), p. 2426, ISSN : 1424-8220, DOI : 10.3390/s17102426.

[60] T. Zhao et Y. Wang, « Market based multi-robot coordination for a cooperative collecting and transportation problem », in : *2013 Proceedings of IEEE Southeastcon*, avr. 2013, p. 1-6, DOI : 10.1109/SECON.2013.6567397.

[61] Guanjie Zheng et al., « DRN : A Deep Reinforcement Learning Framework for News Recommendation », en, in : *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, Lyon, France : ACM Press, 2018, p. 167-176, ISBN : 978-1-4503-5639-8, DOI : 10.1145/3178876.3185994.

[62] Zhenpeng Zhou, Xiaocheng Li et Richard N. Zare, « Optimizing Chemical Reactions with Deep Reinforcement Learning », en, in : *ACS Cent. Sci.* 3.*12* (déc. 2017), p. 1337-1344, ISSN : 2374-7943, 2374-7951, DOI : 10.1021/acscentsci.7b00492.

[63] Zhengxia Zou et al., « Object Detection in 20 Years : A Survey », en, in : *arXiv :1905.05055 [cs]* (mai 2019), arXiv : 1905.05055.

TABLE DES FIGURES

Liste des tableaux

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

Université Bretagne Sud

Titre : Contribution des méthodes d'apprentissage à la distribution de tâches dans un cluster robotique.

Mot clés : Apprentissage par renforcement, deep Q-learning, MRTA, cluster robotique

Résumé : La coordination d'un système multi-robots est une tâche complexe nécessitant l'exécution d'algorithmes de calcul intensif. Dès lors, les ressources calculatoires du robot deviennent une préoccupation majeure puisqu'elles conditionnent la capacité du système à interagir avec l'environnement. Si le recours au cloud robotique permet de s'affranchir de cette difficulté, son utilisation demeure conditionnelle et impraticable dans de nombreux cas.

Une autre solution consiste à organiser le système multi-robots en un cluster robotique où les ressources non-utilisées sont mutualisées afin de paralléliser les tâches fortement calculatoires. Cependant, ces ressources mutualisées s'avèrent limitées et fluctuantes nécessitant la mise en place d'un processus d'allocation dynamique.

Ces travaux de thèse sont consacrés aux apports de l'apprentissage par renforcement dans l'administration d'un cluster robotique évoluant dans un environnement dynamique et incertain. Dans un nouveau contexte appelé MRpTA, deux axes sont étudiés :
Le premier traite du transfert de tâches au sein d'un cluster où des approches classiques sont comparées à une solution utilisant un DQN. Le deuxième axe aborde, dans un contexte de recherche et sauvetage, la question de la distribution des tâches au sein d'un cluster. Le recours à l'apprentissage par renforcement permet d'améliorer considérablement les performances du système.

Title: Contribution of learning methods to task distribution in a robotic cluster.

Keywords: Reinforcement learning, deep Q-learning, MRTA, robotic cluster

Abstract: The multi-robot systems coordination is a complex task requiring the execution of computationaly intensive algorithms. Therefore, the robot's computing resources become a major concern since they condition the system's ability to interact with the environment. While the use of the robotic cloud removes this limitation, its use remains conditional and intractable in many cases.

Another solution consists of organizing the multi-robot system into a robotic cluster where the unused resources are pooled in order to parallelize the highly computational tasks. However, these pooled resources turn out to be limited and fluctuating, requiring the implementation of a dynamic allocation process.

These thesis works are devoted to the contributions of reinforcement learning in the administration of a robotic cluster operating in a dynamic and uncertain environment. In a new context called MRpTA, two axis are studied:
The first deals with the transfer of tasks within a cluster where classical approaches are compared to a DQN solution. The second axis addresses, in a search and rescue context, the question of the distribution of tasks within a cluster. The use of reinforcement learning drastically improves the system performance.