Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Contribution à la résolution des processus de décision markoviens décentralisés

THÈSE

présentée et soutenue publiquement le Date

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Daniel Szer

Composition du jury

Président :	René Schott, Professeur à l'Université Henri-Poincaré Nancy
Rapporteurs :	Eric Hansen, Professeur à la Mississippi State University Abdel-Illah Mouaddib, Professeur à l'Université de Caen
Examinateurs :	Shlomo Zilberstein, Professeur à l'University of Massachusetts Amherst Olivier Sigaud, Professeur à l'Université Paris 6
Directeur :	François Charpillet, Directeur de recherche INRIA

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

Mis en page avec la classe thloria.

Make everything as simple as possible, but not simpler.
- Albert Einstein -

Table des matières

Table des figures

Chapitre 1

Résumé Etendu

1.1 Introduction

1.1.1 Le Problème du Contrôle Décentralisé Optimal

Nous nous intéressons dans ce travail au problème de la planification et de l'apprentissage optimal pour un groupe d'agents coopératifs dans un environnement incertain et partiellement observable. On rencontre de tels problèmes fréquemment dans des domaines comme le routage de paquets dans les réseaux de communication [Alt00], les chaînes d'approvisionnements distribuées [Che99], le contrôle de robots d'intervention pour secourir des populations après une catastrophe [KT01] ou celui de robots d'exploration planétaire [MS01]. Un cadre formel pour décrire la problématique de la décision distribuée a été établi récemment avec le modèle DEC-POMDP [BGIZ02]. Il est fondé sur les processus de décision markoviens (MDPs) et intègre le fait que différents agents agissent dans un environnement stochastique qu'ils n'observent que partiellement. Résoudre ce genre de problèmes est particulièrement difficile, car les informations partielles que possèdent les agents sont en général différentes et potentiellement incohérentes voire contradictoires. De plus, il a été montré que la résolution d'un DEC-POMDP est un problème NEXP-complet à horizon fini [BGIZ02], contrairement au cas POMDP mono-agent, qui lui est PSPACE-dur [PT87]. Trouver des algorithmes efficaces pour résoudre ou approximer des DEC-POMDPs constitue par conséquent un important défi de recherche.

1.1.2 Travaux Existants

Puisque le problème du contrôle optimal décentralisé n'a été formellement défini que très récemment avec les modèles DEC-POMDP [BGIZ02] et MTDP [PT02], ses méthodes de résolution constituent encore un domaine de recherche important. Un premier algorithme général et optimal a été proposé par Hansen et al. pour les problèmes à horizon fini [HBZ04]. Il est basé sur la programmation dynamique et l'élimination itérative de politiques dominantes. Une proposition pour traiter le cas à horizon infini, qui lui est indécidable dans le cas général, peut être trouvée dans [BHZ05]. Cette méthode utilise des méthodes de programmation linéaire pour optimiser des automates finis.

Puisque la complexité du cas général est doublement exponentielle, il est utile de s'intéresser à des classes de problèmes avec des contraintes particulières, pouvant faciliter la résolution. Une première approche a été proposée par Becker et al. [BZLG03]. Celle-ci traite une sous-classe de MDPs décentralisés de manière optimale, à savoir les processus où les agents évoluent de

manière indépendante, mais où seul le comportement collectif est récompensé (*transition independent DEC-MDPs*). Nair et al. exploitent la propriété de connectivité locale entre les agents, c'est-à-dire le fait que les actions d'un agent n'ont un effet direct que sur un nombre limité de voisins, comme c'est le cas par exemple dans les réseaux de communication [NVTY05]. Une synthèse plus exhaustive sur les MDP décentralisés dotés d'une structure particulière peut être trouvée dans [GZ04].

Ils existent en outre plusieurs approches qui se concentrent sur l'approximation de la solution optimale des MDPs décentralisés. Des formalismes d'apprentissage par descente de gradient ont ainsi été proposés par [PKMK00] et par [DBC01]. D'autres approches se fondent sur le concept d'équilibre de Nash, issue de la théorie des jeux, et l'approximation itérative de stratégies par le choix d'une politique de "meilleure réponse" à un ensemble de stratégies déjà existantes [CSC02], [NTY+03], [VNLP02]. L'avantage de ces approches réside dans leur relative simplicité et leur rapidité de convergence, l'inconvénient principal étant la difficulté d'approximer l'erreur que l'on risque de commettre sur la politique optimale.

1.1.3 Apports de la Thèse

Nous allons introduire dans cette thèse plusieurs algorithmes, permettant la résolution optimale ou approchée, par planification ou par apprentissage, du problème du contrôle décentralisé optimal. Nous apports sont plus particulièrement :

- Un algorithme de recherche heuristique optimal pour la résolution de DEC-POMDPs à horizon fini, appelé MAA*.
- L'extension de MAA* aux problèmes à horizon infini, basé sur la recherche de politiques sous forme d'automates finis.
- Un algorithme de programmation dynamique à base de points qui représente une synthèse de la programmation dynamique pour DEC-POMDPs et l'approximation de la fonction de valeur à base de points pour POMDPs.
- Un algorithme d'apprentissage par renforcement distribué pour des problèmes multi-agents coopératifs mais à récompenses individuelles.
- Plusieurs considérations théoriques relatives aux contrôle décentralisé optimal.

Nous présentons finalement des résultats expérimentaux pour valider nos approches par rapport aux algorithmes existants, et nous discutons quelques voies pour des recherches futures.

1.2 Processus de Décision Markovien

Nous introduisons ici le fondement théorique de nos travaux, c'est-à-dire un formalisme mathématique pour la description et la résolution de problèmes de décision séquentiels, les *processus de décision markovien* ou *MDP*, introduits dans les années 60 par Richard Bellman et Ronald Howard [Bel57, How60]. D'un côté, le modèle MDP est très général et assez abstrait pour pouvoir être appliqué à des problématiques variées. D'un autre côté, il reste restreint à des environnements entièrement observables, et le contrôle se fait donc de manière centralisé. Si l'état du système est caché, ou si plusieurs entités contrôlent l'environnement en même temps, le modèle MDP doit être étendu, aux modèles POMDP (MDP partiellement observable) et DEC-POMDP (POMDP décentralisé) respectivement. Notons aussi qu'une introduction exhaustive sur les MDP en général peut être trouvée dans l'ouvrage de Puterman [Put94].

1.2.1 MDPs

Nous commençons par introduire le formalisme MDP, ses propriétés charactéristiques, ainsi que quelques algorithmes fondamentaux permettant sa résolution.

1.2.1.1 Formalisme

Definition 1.2.1 (MDP). *Un* MDP *est défini par* $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$*, où*
- $\mathcal{S}$ *est un ensemble fini d'états* $s \in \mathcal{S}$*,*
- $\mathcal{A}$ *est un ensemble fini d'actions* $a \in \mathcal{A}$*,*
- $\mathcal{T}(s, a, s')$ *est la probabilité de transition du système de l'état* s *vers l'état* s' *après exécution de l'action* a*,*
- $\mathcal{R}(s, a)$ *est la récompense produite par le système lorsque action* a *est exécutée dans l'état* s*.*

La propriété fondamentale du formalisme MDP est la *propriété de Markov*. Elle garanti que la probabilité de transition d'un état s vers un état s' sous l'effet d'une action a est indépendante du passé :

$$P(s_{t+1} = s' | s_0, a_0, s_1, a_1, \ldots, s_t, a_t) = P(s_{t+1} = s' | s_t, a_t) \tag{1.1}$$

Par conséquent, l'état courant constitue toujours une information suffisante pour la prédiction de l'évolution du système.

Exécuter un MDP revient à choisir en boucle une action a à effectuer, et à observer la transition de l'état courant s du système vers son nouvel état s'. On appelle *politique* π une loi de décision qui détermine à chaque instant t et pour chaque état s l'action à effectuer.

On appelle *problème de décision markovien* un MDP avec un *critère de performance* associé. Le critère de performance précise comment on doit évaluer la qualité d'une politique pour un MDP donné. Une fois que le critère de performance est fixé, on peut s'intéresser à la politique qui optimise l'évaluation de ce critère. La recherche d'une telle politique constitue l'enjeu principal du problème de décision markovien.

Pour des problèmes à horizon fini T, un des critères de performance les plus utilisés est l'espérance de la somme des récompenses accumulées à partir de l'état initial s_0 :

$$E\left[\sum_{t=0}^{T-1} \mathcal{R}\left(s_t, a_t\right) \;\middle|\; s_0\right] \tag{1.2}$$

Pour des problèmes à horizon infini, utiliser le même critère de performance pose le problème de l'accumulation non bornée de récompenses. C'est pour cela que l'on introduit en général un facteur de décompte γ, ce qui donne le critère suivant :

$$E\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}\left(s_t, a_t\right) \;\middle|\; s_0\right] \qquad \text{avec} \qquad 0 \leq \gamma < 1 \tag{1.3}$$

Dès qu'un critère de performance est associé à un MDP, on peut évaluer la valeur d'une politique en établissant sa *fonction de valeur* V, c'est-à-dire une fonction qui retourne pour chaque état la valeur du critère de performance. On obtient ainsi

$$V(s, \pi) = E\left[\sum_{t=0}^{T-1} \mathcal{R}\left(s_t, a_t\right) \;\middle|\; s_0 = s\right] \qquad \text{à horizon fini} \tag{1.4}$$

ou bien

$$V(s,\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}\left(s_t, a_t\right) \;\middle|\; s_0 = s\right] \qquad \text{à horizon infini.} \tag{1.5}$$

Une fois que le critère de performance a été établi, on peut s'intéresser à résoudre le problème de décision markovien, c'est-à-dire à trouver la politique qui maximise la fonction de valeur pour un état initial s_0 donné. On parlera alors de *politique optimale* que l'on notera π^* :

$$\pi^* = \arg\max_{\pi} V(s_0, \pi) \tag{1.6}$$

En général, plusieurs politiques optimales peuvent exister pour un même MDP. Dans ce qui suit, nous allons nous limiter à des algorithmes permettant à en déterminer une.

1.2.1.2 Résolution par Planification

On peut montrer que la politique optimale pour des MDPs à horizon fini et pour le critère de performance considéré ci-dessus est déterministe, indépendante du passé mais pas stationnaire [Put94]. Une politique optimale choisira donc toujours la même action pour une configuration donnée, mais cette action dépendra du moment d'exécution. On dénotera $\pi = (\pi_T, \pi_{T-1}, \ldots, \pi_1)$ une telle politique et $\pi_t(s) = a$ l'action à effectuer dans l'état s au moment t de l'exécution. Établir la fonction de valeur d'une telle politique peut se faire directement en déroulant la fonction de valeur, et en déterminant explicitement les espérances de récompenses pour chaque instant t :

$$V(s,\pi_t) = \mathcal{R}(s,\pi_t(s)) + \sum_{s' \in S} \mathcal{T}(s,\pi_t(s),s') V(s',\pi_{t-1}) \tag{1.7}$$

Puisque le calcul de la fonction de valeur au moment t nécessite la connaissance de la fonction de valeur au moment $(t-1)$, on commence par l'étape finale, pour ensuite rétropropagager les valeurs pour le calcul des espérances au moment précédent. Nous allons présenter un moyen pour faire cela avec l'algorithme 1.

Algorithm 1 Évaluation d'une politique à horizon fini - `EvaluationPolitique()`

Require: Un MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ et une politique π_T
Ensure: La fonction de valeur V pour la politique π_T

```
V(s, π_0) ← 0,    (∀s ∈ S)
for t = 1 to t = T do
    for all s ∈ S do
        V(s, π_t) ← R(s, π_t(s)) + Σ_{s'∈S} T(s, π_t(s), s') V(s', π_{t-1})
    end for
end for
```

Déterminer une politique optimale peut faire appel à la procédure `EvaluationPolitique()` sur l'ensemble des politiaues possibles pour garder ensuite le meilleur choix. Il s'agit alors d'une recherche exhaustive dans l'espace des politiques. Richard Bellman a introduit une approche plus efficace pour déterminer une politique optimale. Son *principe d'optimalité* stipule que la politique optimale $\pi^* = (\pi_T^*, \pi_{T-1}^*, \ldots, \pi_1^*)$ pour l'horizon T contient forcément une politique optimale pour l'horizon $(T-1)$, à savoir la politique $\pi^* = (\pi_{T-1}^*, \ldots, \pi_1^*)$. De manière plus générale, elle contient des sous-politiques optimales pour tout horizon $t \leq T$. Le principe garanti

donc que le calcul d'une politique optimale pour un horizon T peut utiliser la solution pour l'horizon $(T-1)$ du même MDP. Il suffit alors de trouver la fonction de décision optimale pour le début, à savoir la fonction π_1^*. Trouver une politique optimale pour un horizon T nécessite que ce calcul soit répété T fois, ce qui signifie qu'un nombre total de $T \cdot |\mathcal{A}|^{|\mathcal{S}|}$ politiques doit être considéré. La réduction considérable du nombre d'évaluations par rapport à la méthode naïve en réutilisant des solutions intermédiaires constitue le coeur de ce qu'on appelle la *programmation dynamique*. L'algorithme 2, appelé *backward induction*, permet de calculer la politique optimale d'un MDP à horizon fini par le moyen de la programmation dynamique. Le calcul d'une poli-

Algorithm 2 Backward induction - `BackwardInduction()`

Require: Un MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ et un horizon T
Ensure: Une politique optimale π_T^* pour l'horizon T
$V(s, \pi_0) \leftarrow 0, \quad (\forall s \in \mathcal{S})$
for $t = 1$ **to** $t = T$ **do**
  for all $s \in \mathcal{S}$ **do**
    $\pi_t(s) \leftarrow \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') V(s', \pi_{t-1}) \right]$
    $V(s, \pi_t) \leftarrow \mathcal{R}(s, \pi_t(s)) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi_t(s), s') V(s', \pi_{t-1})$
  end for
end for

tique optimale peut en principe être étendue directement aux problèmes à horizon infini, avec la seule contrainte que le nombre de fonctions de décision soit possiblement infiniment grand. Fort heureusement, on peut montrer que, pour le critère de performance choisi pour l'horizon infini, il suffit de se limiter à des politiques déterministes, markoviennes et stationnaires [Put94]. Le fait que la politique optimale soit stationnaire signifie que toutes les fonctions de décision sont identiques $\pi_1^* = \pi_2^* = \cdots = \pi_\infty^* = \pi^*$. Ainsi, il n'existe plus qu'une seule fonction de valeur. L'algorithme 3, appelé *itération de la valeur*, détermine une approximation de cette fonction de valeur optimale V^*. Une politique optimale peut ensuite être déduite de la fonction de valeur

Algorithm 3 Itération de la valeur - `IterationValeur()`

Require: Un MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ et une borne d'erreur ϵ
Ensure: Une approximation ϵ-proche de la fonction de valeur optimale
$V^0(s) \leftarrow 0, \quad (\forall s \in \mathcal{S})$
$n \leftarrow 0$
repeat
  for $s \in \mathcal{S}$ **do**
    $V^{n+1}(s) \leftarrow \max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \gamma V^n(s') \right]$
  end for
  $n \leftarrow n + 1$
until $||V^n - V^{n-1}|| \leq \epsilon$

par simple *one-step lookahead* :

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \gamma V^*(s') \right] \tag{1.8}$$

1.2.1.3 Résolution par Apprentissage

Le paragraphe précédent introduisait des techniques de planification hors-ligne, où le modèle exact du MDP, c'est-à-dire sa fonction de transition $\mathcal{T}$ et sa fonction de récompense $\mathcal{R}$, était connue. L'exécution de la politique optimale se faisait alors indépendamment du processus de planification. L'apprentissage d'une politique est un processus en-ligne, c'est-à-dire que l'agent doit en même temps apprendre à connaître l'environnement et déterminer une politique optimale pour le contrôler. La boucle de contrôle de l'agent est alors la suivante : l'agent perçoit l'état courant s, il choisit une action a, le système fait la transition vers un nouvel état s' et l'agent reçoit une récompense $\mathcal{R}(s,a)$ qui lui indique l'utilité locale de cette transition. Le problème est alors de déterminer une politique optimale pendant que l'exploration de l'environnement est encore en cours.

Les problèmes d'apprentissage sont souvent des problèmes à horizon infini, et un des algorithmes les plus répandus est celui du *Q-learning*, introduit par Watkins [Wat89, WD92]. Le Q-learning est la version en-ligne de l'algorithme de l'itération de la valeur. Il est fondé sur l'évaluation des couples état-action, et non directement sur la fonction de valeur. La fonction $Q(s,a)$ représente la valeur espérée lorsque l'action a est exécutée en s et qu'une politique optimale est suivie à partir de l'état prochain s'. La fonction Q et la fonction de valeur V sont étroitement liées :

$$V(s) = \max_{a \in \mathcal{A}} Q(s,a) \tag{1.9}$$

Watkins a pu démontrer que la mise à jour suivante, effectuée après chaque exécution d'une action, garanti la convergence vers la fonction Q optimale, sous réserve que chaque couple état-action soit visité infiniment souvent et que le facteur α vérifie certaines propriétés basiques ($0 \leq \alpha, \sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$) :

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a') \right] \tag{1.10}$$

L'approche du Q-learning est décrite par l'algorithme 4.

Algorithm 4 Q-learning - `QLearning()`

Require: Un MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$
Ensure: Un approximation de la fonction Q optimale
1: $Q(s,a) \leftarrow 0, \quad (\forall s \in \mathcal{S})(\forall a \in \mathcal{A})$
2: **loop**
3: Exécuter une action a
4: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a') \right]$
5: **end loop**

1.2.2 POMDPs

La loi de décision dans un MDP est toujours fondée sur l'état réel du système. Malheureusement, il existe des cas où cet état n'est pas accessible. Tout ce dont dispose le contrôleur lors de l'exécution est un signal ou une observation bruitée, à l'aide duquel il peut au mieux essayer d'inférer la vraie configuration du système. De tels systèmes peuvent être modélisés à l'aide de ce qu'on appelle les *processus de décision markovien partiellement observables* où POMDPs.

1.2.2.1 Formalisme

Definition 1.2.2 (POMDP). *Un POMDP est défini par* $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$*, où*

- $\mathcal{S}$ *est un ensemble fini d'états* $s \in \mathcal{S}$*,*
- $\mathcal{A}$ *est un ensemble fini d'actions* $a \in \mathcal{A}$*,*
- $\mathcal{T}(s, a, s')$ *est la probabilité de transition du système de l'état* s *vers l'état* s' *après exécution de l'action* a*,*
- $\mathcal{R}(s, a)$ *est la récompense produite par le système lorsque l'action* a *est exécutée dans l'état* s*,*
- Ω *est un ensemble fini d'observations* $o \in \Omega$*,*
- $\mathcal{O}(s, a, o, s')$ *est la probabilité de l'observation* o *lors de la transition du système de l'état* s *vers l'état* s' *après exécution de l'action* a*.*

Le processus restreint à $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *est souvent appelé* MDP sous-jacent *au POMDP.*

A la différence de l'état courant, qui lui est une information suffisante pour le contrôle optimal d'un MDP, l'observation courante ne vérifie pas nécessairement la propriété de Markov dans le cas d'un POMDP. Ceci veut dire que les méthodes de résolution et les équations d'optimalité qui sont valables pour les états d'un MDP ne peuvent pas être appliqués directement aux observations d'un POMDP. Nous allons montrer maintenant qu'il est toutefois possible de se ramener à une autre description du système où la propriété de Markov est à nouveau vérifiée. On peut en effet constater que la probabilité du système de se trouver dans un état s au moment t de l'exécution dépend uniquement de la distribution de probabilités sur les états au moment précédent. Pour cela, nous posons $\boldsymbol{b_t}(s)$ la probabilité du système de se trouver dans l'état s au moment t de l'exécution :

$$\boldsymbol{b_t}(s) = P(s_t = s | s_0, a_0, o_0, \ldots, a_{t-1}, o_{t-1}) \tag{1.11}$$

Le vecteur $\boldsymbol{b}$

$$\boldsymbol{b_t} = \langle \boldsymbol{b_t}(s_1), \boldsymbol{b_t}(s_2), \ldots, \boldsymbol{b_t}(s_{|\mathcal{S}|}) \rangle \tag{1.12}$$

est souvent appelé *état de croyance* ou *belief state*. Sa mise-à-jour après l'exécution d'une action a et la perception d'une observation o peut être dérivée de la formule de Bayes

$$\boldsymbol{b_{t+1}}(s') = \frac{\sum_{s \in \mathcal{S}} \boldsymbol{b_t}(s) \Big[\mathcal{T}(s, a_t, s') \mathcal{O}(s, a_t, o_t, s') \Big]}{P(o | \boldsymbol{b}, a)} \tag{1.13}$$

où $P(o|\boldsymbol{b}, a)$ est un facteur de normalisation.

Il a été montré par Astrom [Ast65] que l'état de croyance constitue une information suffisante pour représenter le passé du système. Il est donc possible de considérer un POMDP comme un MDP à états continus, le *belief-state MDP*. L'obstacle principal à sa résolution réside dans la continuité de son espace d'états. Un opérateur de type itération de valeur devrait donc être appliqué une infinité de fois pour établir une fonction de valeur. Il a cependant été montré par Smallwood et Sondik [SS73] que la fonction de valeur à l'itération i peut toujours être représentée par un ensemble fini de paramètres. En effet, si V_i est une fonction de valeur convexe et linéaire par morceaux, alors la fonction V_{i+1} est aussi convexe est linéaire par morceaux. On peut montrer en particulier que la fonction de valeur à horizon 1 est forcément convexe et linéaire par morceaux. Ceci garanti, avec la propriété précédente, que la fonction de valeur pour un horizon fini est toujours représentable par des moyens finis. Le théorème de Smallwood et Sondik représente le fondement essentiel pour la résolution d'un POMDP.

Nous avons constaté que l'observation courante ne vérifie plus la propriété de Markov dans le cas POMDP. A la différence du MDP, une politique optimale pour un POMDP est donc histoire dépendante. Elle peut être représentée sous forme d'un arbre de décision q, encore appelé *arbre de politique* [KLC98]. Les nœuds de l'arbre représentent les actions à effectuer, et le parcours de l'arbre est choisi en fonction des observations reçues. Nous appelons $\alpha(q)$ l'action associée à la racine de l'arbre, $q(o)$ le sous-arbre associé à l'observation o, et $\lambda(q, i)$ la i'ième feuille de l'arbre q. Un exemple d'un arbre de politique à horizon 3 est montré en figure 1.1.

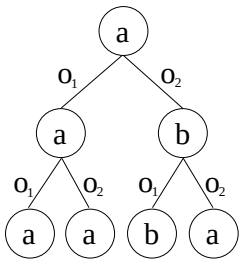

FIG. 1.1 – Un exemple d'un arbre de politique à horizon 3, pour un problème à 2 observations o_1, o_2, et à 2 actions a, b.

1.2.2.2 Résolution par Programmation Dynamique

La fonction de valeur d'une seule politique peut être représentée sous forme d'un vecteur $|\mathcal{S}|$-dimensionnel $\boldsymbol{\alpha} = \langle V(s_1, q), \ldots, V(s_{|\mathcal{S}|}, q)\rangle$. La valeur de tout état de croyance intermédiaire peut ensuite être obtenue par simple interpolation linéaire :

$$V(\boldsymbol{b}, q) = \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V(s, q) \tag{1.14}$$

Si un ensemble de politiques Q est donné, alors la valeur de tout état de croyance $\boldsymbol{b}$ peut être déterminé par maximisation sur les politiques disponibles :

$$V(\boldsymbol{b}) = \max_{q \in Q} V(\boldsymbol{b}, q) = \max_{q \in Q} \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V(s, q) \tag{1.15}$$

Conformément au théorème de Smallwood et Sondik, il y a un ensemble fini de politiques possibles, et donc un nombre fini de vecteurs $\boldsymbol{\alpha}$ à considérer pour établir la fonction de valeur. Un exemple d'une telle fonction de valeur est montré en Figure 1.2. Pour chaque état de croyance $\boldsymbol{b}$, il y a un vecteur $\boldsymbol{\alpha}$, et donc un arbre de politique associé, qui est optimal. La question fondamentale pour la résolution optimale d'un POMDP est alors la suivante : est-ce qu'il est possible d'identifier des politiques qui sont sous-optimales sur l'ensemble de l'espace des croyances ? Ces politiques peuvent alors être supprimées.

Une politique qui est optimale en un état de croyance donné est appelé la politique *dominante en cet état*. De la même manière, une politique qui est sous-optimale est appelée politique *dominée en cet état*. Ce qui nous intéresse c'est d'identifier les politiques qui sont toujours dominées par au moins une autre politique, et ceci pour n'importe quel état de croyance. Une telle politique est alors appelée *politique entièrement dominée* ou simplement *politique dominée*. Le vecteur $\boldsymbol{\alpha}$

associé à cette politique est appelé *vecteur dominé*. Tout vecteur qui n'est pas entièrement dominé est appelé *vecteur utile*. Un exemple d'une fonction de valeur avec deux vecteurs utiles et un vecteur dominé est montré en figure 1.2. Il est important d'insister sur le fait qu'un vecteur

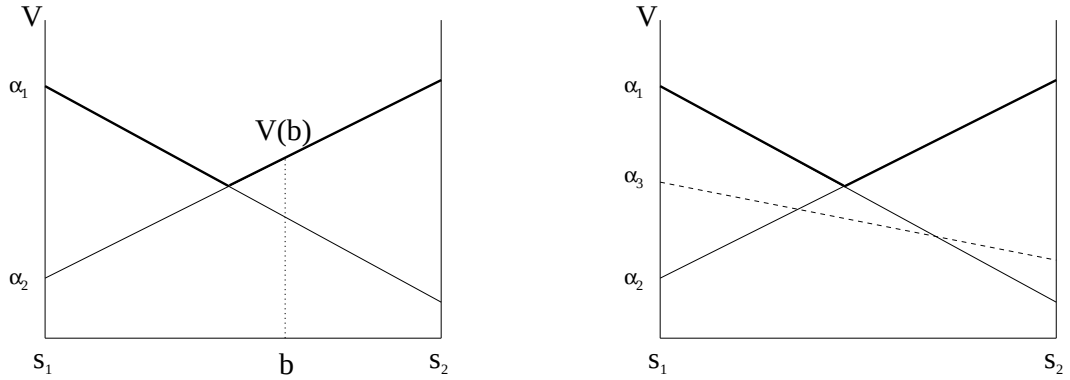

FIG. 1.2 – La fonction de valeur d'un POMDP à deux états s_1 et s_2. L'action associée au vecteur $\boldsymbol{\alpha_2}$ correspond à l'action optimale en $\boldsymbol{b}$. Le vecteur $\boldsymbol{\alpha_3}$ est entièrement dominé et peut être supprimé sans affecter la représentation de la fonction de valeur optimale.

$\boldsymbol{\alpha}$ dominé peut être supprimé sans affecter la représentation de la fonction de valeur optimale. Il en est de même pour la politique associée. On peut en outre montrer qu'une politique dominée ne peut jamais faire partie d'une politique optimale, c'est-à-dire qu'un arbre de politique dominé ne peut jamais apparaître comme sous-abre dans une politique optimale. Déterminer et supprimer les vecteurs dominés constitue l'effort majeur dans la programmation dynamique pour les POMDPs. Un vecteur $\boldsymbol{\alpha}$ est dominé, si la politique q associée vérifie

$$(\forall \boldsymbol{b})(\exists \tilde{q} \neq q) \qquad \text{tel que} \qquad V(\boldsymbol{b}, q) \leq V(\boldsymbol{b}, \tilde{q}) \tag{1.16}$$

Cette propriété peut être vérifié par la résolution d'un programme linéaire.

Nous avons vu qu'il existe une correspondance entre l'ensemble des vecteurs $\boldsymbol{\alpha} \in \Gamma$ qui constituent la fonction de valeur à horizon t, et l'ensemble des arbres de politique $q \in Q$ à profondeur t. La construction par programmation dynamique de la fonction de valeur optimale est donc équivalente à la construction d'un ensemble de politiques utiles. Comme c'est le cas pour les MDPs, cette construction se fait horizon par horizon. Supposons l'existence d'une fonction de valeur optimale pour l'horizon t, représentée par un ensemble de vecteurs Γ^t, et donc par un ensemble d'arbres de politique Q^t. Au début de l'algorithme, cet ensemble sera l'ensemble des politiques à horizon 1, c'est-à-dire l'ensemble $\mathcal{A}$ des actions simples. Créer la fonction de valeur pour l'horizon $(t+1)$ revient à générer d'abord l'ensemble des politiques à horizon $(t+1)$. Pour construire un nouvel arbre de politique q^{t+1}, il faut choisir une action $a \in \mathcal{A}$ pour l'affecter à la racine, ainsi que $|\Omega|$ politiques $q_i^t \in Q^t$ pour les associer aux observations. Il y a donc un nombre total de $|\mathcal{A}||Q^t|^{|\Omega|}$ politiques à générer, et nous allons appeler *génération exhaustive* tout processus qui construit l'ensemble de ces politiques en un coup. Ensuite, toute politique entièrement dominée est supprimée de l'ensemble Γ^{t+1}.

Une première approche de programmation dynamique pour POMDPs est résumée dans l'algorithme 5. Il est important de noter que la construction des arbres de politique, et donc de la fonction de valeur, se fait "de bas en haut", c'est-à-dire que l'on génère d'abord l'ensemble

des arbres possiblement utiles au dernier pas de l'exécution - donc les feuilles de la future politique. Lors de chaque itération, on construit un ensemble de politiques qui contiennent comme sous-arbres les politiques utiles de l'itération précédente. Il s'agit en effet du procédé analogue à l'algorithme *backward induction* dans le cas du MDP.

Algorithm 5 Programmation dynamique pour POMDPs - `ProgrammationDynamiquePOMDPs()`

Require: Un POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$ et un T
Ensure: Un ensemble de politiques optimales pour l'horizon T

Initialiser $Q^1 \leftarrow \mathcal{A}$
for $t = 2$ **to** $t = T$ **do**
  /* Génération exhaustive de politiques */
  for all action $a \in \mathcal{A}$ **do**
    for all choix de $1 \leq i \leq |\Omega|$ politiques $q_i^{t-1} \in Q^{t-1}$ **do**
      Construire une nouvelle politique q^t telle que :
      $\alpha(q^t) \leftarrow a$
      $q^t(o_i) \leftarrow q_i^{t-1}, \qquad (\forall o_i \in \Omega)$
      $Q^t \leftarrow Q^t \bigcup \{q^t\}$
    end for
  end for
  /* Élagage */
  Supprimer toute politique $q \in Q^t$ si $(\forall \boldsymbol{b})(\exists \tilde{q} \neq q)$ telle que $V(\boldsymbol{b}, q) \leq V(\boldsymbol{b}, \tilde{q})$
end for

1.2.2.3 Résolution par Recherche Heuristique

Construire un arbre de politique optimal peut aussi être interprété comme la recherche d'un chemin conditionnel dans l'espace des politiques possibles. L'arbre de recherche POMDP est un exemple de ce qu'on appelle un *arbre ET/OU* [Pea90], et la construction de la politique optimale se fait alors "de haut en bas". Pour chaque nœud d'action, il faut considérer toutes les observations possibles, c'est-à-dire $|\Omega|$ nœuds fils (partie ET). Pour chaque observation, il faut ensuite choisir un seul nœud fils (partie OU). Un arbre de recherche ET/OU pour un POMDP est montré à titre d'exemple en figure 1.3. Trouver le chemin conditionnel optimal dans un arbre ET/OU peut se faire par recherche heuristique, et plus précisément par l'algorithme A* [Nil80, Pea90]. A* est un algorithme de recherche du type meilleur-d'abord, combiné avec une fonction heuristique pour sélectionner le meilleur nœud courant. Évaluer une feuille λ de profondeur t dans l'arbre de recherche revient à calculer la fonction

$$F^T(\boldsymbol{b_0}, \lambda) = G^t(\boldsymbol{b_0}, \lambda) + H^{T-t}(\boldsymbol{b_0}, \lambda) \tag{1.17}$$

où $\boldsymbol{b_0}$ est l'état de croyance initial, G est la valeur exacte du chemin de la racine jusqu'à la feuille λ, H est une estimation optimiste de la valeur du chemin restant à partir de λ et T est l'horizon du problème [Pea90]. Chaque feuille λ de l'arbre de recherche peut être identifiée de manière unique par une séquence d'actions et d'observations $\lambda \equiv (a_1, o_1, \ldots, a_{t-1}, o_{t-1}, a_t)$, et nous notons $\boldsymbol{b}_{\boldsymbol{b_0},\lambda}$ l'état de croyance associé à l'historique de la feuille λ pour l'état de croyance initial $\boldsymbol{b_0}$.

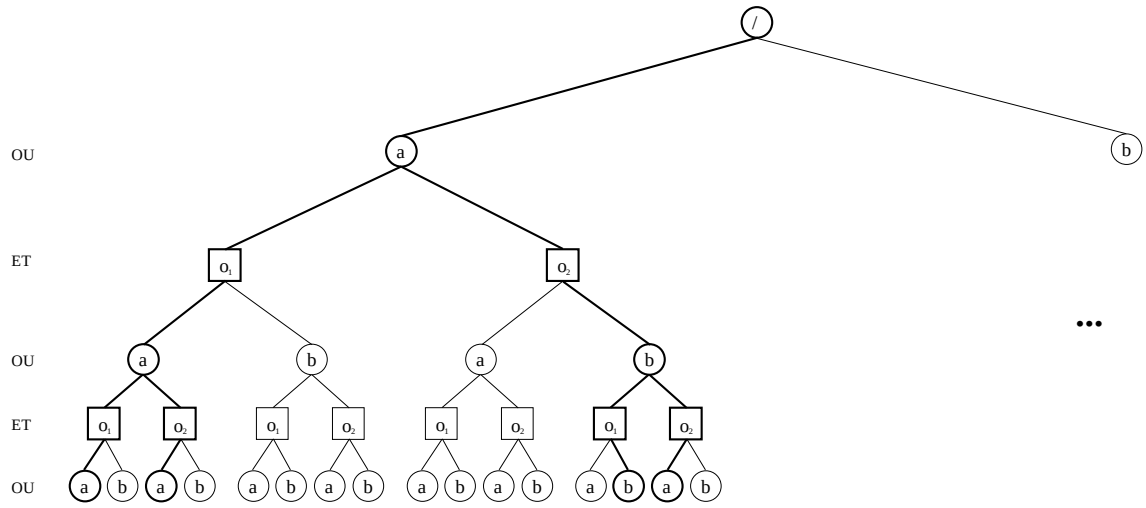

FIG. 1.3 – Une partie d'un arbre de recherche de type ET/OU. Les nœuds a et b sont des nœuds de type OU qui représentent les actions, alors que les nœuds o_1 et o_2 sont des nœuds de type ET qui représentent les observations. La solution indiqué en gras représente la même politique que celle montré figure 1.1

Déterminer G Évaluer le chemin de la racine de l'arbre de recherche jusqu'à la feuille λ revient à calculer la valeur d'une chaîne de Markov :

$$V(s,(a_1,o_1,a_2,o_2,\ldots,a_t)) = \mathcal{R}(s,a_1) + \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a_1,s')\mathcal{O}(s,a_1,o_1,s')V(s',(a_2,o_2,\ldots,a_t)) \tag{1.18}$$

La fonction G n'est alors rien d'autre que la somme pondérée de ces évaluations :

$$G^t(\boldsymbol{b_0},\lambda) = \sum_{s\in\mathcal{S}} \boldsymbol{b_0}(s)V(s,\lambda) \tag{1.19}$$

Déterminer H Tandis que le chemin de la racine jusqu'à la feuille λ peut être évalué explicitement, l'intérêt de l'heuristique H est d'approximer efficacement la valeur du chemin optimal restant à partir de λ. La garantie de convergence de $A*$ nécessite que H surestime la valeur de ce chemin. Une fonction qui vérifie cette propriété est dite *admissible*. Établir une fonction d'estimation optimiste signifie trouver une borne supérieure à la fonction de valeur d'un POMDP. Une telle borne peut facilement être déterminée en faisant appel au MDP sous-jacent. On peut montrer en effet que

$$V^{T-t}_{POMDP}(\boldsymbol{b}) \leq \sum_{s\in\mathcal{S}} \boldsymbol{b}(s)V^{T-t}_{MDP}(s) \tag{1.20}$$

où V^{T-t}_{POMDP} et V^{T-t}_{MDP} sont respectivement les fonctions de valeur du POMDP et du MDP sous-jacent correspondant [LCK95, Hau00]. La fonction heuristique H sur l'horizon restant $(T-t)$ définie comme

$$H^{T-t}(\boldsymbol{b_0},\lambda) = H^{T-t}(\boldsymbol{b}_{\boldsymbol{b_0},\lambda}) = \sum_{s\in\mathcal{S}} \boldsymbol{b}_{\boldsymbol{b_0},\lambda}(s)V^{T-t}_{MDP}(s) \tag{1.21}$$

est donc admissible par définition. Ils existent d'autres bornes supérieures à la fonction de valeur d'un POMDP, et un résumé étendu peut être trouvé dans [Hau00]. Chacune d'elle peut constituer une fonction heuristique admissible pour guider la recherche de A*.

L'application des méthodes de recherche heuristique pour la résolution de POMDPs a été étudiée notamment par [Was96, GB98], et l'algorithme 6 présente une version de A* pour des problèmes à horizon fini avec une fonction heuristique basée sur le MDP sous-jacent et la fonction d'évaluation suivante :

$$\overline{V}^T(\boldsymbol{b}, \lambda) = \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V(s, \lambda) + \sum_{s \in \mathcal{S}} \boldsymbol{b'}_{\boldsymbol{b},\lambda}(s) V_{MDP}^{T-t}(s) \tag{1.22}$$

Algorithm 6 Recherche heuristique pour POMDPs - `POMDPAEtoile()`

Require: Un POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$ et un horizon T
Ensure: Une politique optimale pour l'état de croyance initial $\boldsymbol{b_0}$
1: Initialiser la liste OPEN D avec un nœud vide $\emptyset$
2: **repeat**
3: Choisir $\lambda \in D$ tel que$(\forall \lambda' \neq \lambda) : \overline{V}(\boldsymbol{b_0}, \lambda') \leq \overline{V}(\boldsymbol{b_0}, \lambda)$
4: Construire une nouvelle politique q telle que :
5: La racine de q est identique avec $\lambda : \alpha(q) \leftarrow \lambda$
6: **for all** $o \in \Omega$ **do**
7: /* Construire un fils */
8: $q(o) \leftarrow o$
9: **for all** $a \in \mathcal{A}$ **do**
10: /* Construire un arrière-fils */
11: $q(o)(a) \leftarrow a$
12: **end for**
13: **end for**
14: Remplacer λ avec q
15: **until** la meilleure politique courante λ est de profondeur T

1.2.3 DEC-POMDPs

Le modèle DEC-POMDP est l'extension naturelle du formalisme POMDP à des problèmes où la prise de décision est distribuée. Ceci veut dire que plusieurs entités, aussi appelés *agents*, influencent le système en même temps. Chaque agent choisit son action en fonction d'une politique locale, mais l'évolution du système dépend de l'action jointe de tous les agents. Le modèle DEC-POMDP est donc prédestiné à la description de systèmes multi-agents. Il a été introduit par Bernstein et al. dans [BZI00], et il constitue le formalisme théorique de nos travaux.

1.2.3.1 Formalisme

Definition 1.2.3 (DEC-POMDP). *Un DEC-POMDP est défini par un tuple* $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$, *où*

- $\mathcal{S}$ *est un ensemble fini d'états* $s \in \mathcal{S}$,
- $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *est un ensemble fini d'actions jointes* $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$, *et* $\mathcal{A}_i$ *dénote l'ensemble des actions* a_i *pour l'agent* i,
- $\mathcal{T}(s, \boldsymbol{a}, s')$ *est la probabilité de transition du système de l'état* s *vers l'état* s' *après exécution de l'action jointe* $\boldsymbol{a}$,
- $\mathcal{R}(s, \boldsymbol{a})$ *est la récompense produite par le système lorsque l'action jointe* $\boldsymbol{a}$ *est exécutée dans l'état* s,

- $\boldsymbol{\Omega} = \Omega_1 \times \cdots \times \Omega_n$ *est un ensemble fini d'observations jointes* $\boldsymbol{o} \in \boldsymbol{\Omega}$*, et* Ω_i *dénote l'ensemble d'observations* o_i *pour l'agent* i*,*
- $\mathcal{O}(s, \boldsymbol{a}, \boldsymbol{o}, s')$ *est la probabilité de l'observation jointe* $\boldsymbol{o}$ *lors de la transition du système de l'état* s *vers l'état* s' *après exécution de l'action joint* $\boldsymbol{a}$*.*

Le processus qui considère les actions jointes comme actions atomiques, et qui ne permet donc pas une exécution décentralisée, est appelé POMDP sous-jacent *au DEC-POMDP.*

Résoudre un DEC-POMDP revient à trouver une politique locale par agent, c'est-à-dire une *politique jointe* pour l'équipe, telle que son exécution distribuée mais synchrone maximise l'espérance des récompenses accumulées par le système. Il n'y a pas de récompenses individuelles, ce qui signifie implicitement que les agents sont coopératifs.

Établir une fonction de valeur dans le cas d'un DEC-POMDP nécessite la redéfinition du concept d'état de croyance. Tout d'abord, l'*état de croyance multi-agent* devient subjectif, et chaque agent possède une autre croyance sur le système. Ensuite, l'état de croyance multi-agent ne consiste pas uniquement en une distribution de probabilités sur l'état sous-jacent du système, mais doit en outre prendre en compte le comportement futur du groupe d'agents. Ceci peut être illustré par l'exemple suivant. Supposons que deux agents se trouvent des deux côtés d'une rivière. Il y a deux ponts a et a, et le but de chaque agent est de traverser la rivière. Si les deux agents choisissent le même pont pour traverser la rivière, il y a collision. Il y a donc deux actions jointes optimales, l'action $\langle a, b\rangle$ et l'action $\langle b, a\rangle$. Toutefois, il n'est pas possible de déterminer par exemple la valeur de l'action a toute seule pour le premier agent : si le deuxième agent choisi l'action b, l'action a est bénéfique, mais si le deuxième agent choisi également l'action a, alors il y a collision et la valeur de l'action a est par conséquent négative. Pour déterminer la valeur d'une politique dans un cadre multi-agent, il faut donc prendre en compte le comportement futur des autres agents. Si Q_i dénote l'ensemble de politiques probables de l'agent i, et $\boldsymbol{Q_{-i}} = \langle Q_1, \ldots, Q_{i-1}, Q_{i+1}, \ldots, Q_n\rangle$ les ensembles des politiques probables de tous les agents sauf l'agent i, alors un état de croyance multi-agent $\boldsymbol{b}_i$ pour un agent i est une distribution de probabilités sur $\mathcal{S}$ et sur $\boldsymbol{Q_{-i}}$: $\boldsymbol{b}_i \in \Delta(\mathcal{S} \times \boldsymbol{Q_{-i}})$.

La définition d'un espace de croyance multi-agent permet d'étendre le concept de fonction de valeur aux DEC-POMDP. On note V_i la *fonction de valeur locale* de l'agent i. Si $\boldsymbol{q}_{-i}$ est la politique jointe pour tous les agents sauf l'agent i, et si $\langle \boldsymbol{q}_{-i}, q_i\rangle$ dénote une politique jointe complète, alors la valeur de la politique q_i en l'état de croyance $\boldsymbol{b}_i$ peut être déterminée comme suit :

$$V_i(\boldsymbol{b}_i, q_i) = \sum_{s \in \mathcal{S}} \sum_{\boldsymbol{q}_{-i} \in \boldsymbol{Q_{-i}}} \boldsymbol{b}_i(s, \boldsymbol{q}_{-i}) V(s, \langle \boldsymbol{q}_{-i}, q_i\rangle) \tag{1.23}$$

Il est important de souligner que la valeur d'une politique locale ne peut pas être évaluée sans prendre en compte le comportement des autres agents : une seule et même action peut être à la fois très bénéfique où très néfaste en fonction du choix des actions des autres membres de l'équipe d'agents. Ceci est vrai à la fois pour des systèmes coopératifs et pour des systèmes compétitifs, et a été souligné plus particulièrement en théorie des jeux, où la notion de politique de *réponse optimale* au comportement d'un groupe a été introduite. Une politique $BR_i(\boldsymbol{b}_i, Q_i)$ est la réponse optimale de l'agent i dans l'état de croyance $\boldsymbol{b}_i$, si

$$BR_i(\boldsymbol{b}_i, Q_i) = \arg\max_{q_i \in Q_i} V_i(\boldsymbol{b}_i, q_i) \tag{1.24}$$

où Q_i dénote l'ensemble de politiques parmi lesquelles l'agent i peut choisir. Déterminer la politique de réponse optimale de l'agent i nécessite que tout agent $j \neq i$ ait déjà choisi sa politique locale. Déterminer la politique locale optimale d'un agent j par un mécanisme de réponse optimale en revanche nécessite que l'agent i soit déjà en possession de sa politique locale - le raisonnement devient donc circulaire. En général, il n'est pas possible de déterminer une politique jointe optimale par n calculs de politiques de réponse optimale. On peut en revanche étendre le concept de politique utile et de politique dominée au cas multi-agent. Ainsi, une politique $q_i \in Q_i$ est dite *dominée* si, pour tout état de croyance multi-agent $\boldsymbol{b}_i$, il existe encore au moins une autre politique $\tilde{q}_i$ qui est au moins aussi performante :

$$(\forall \boldsymbol{b}_i)(\exists \tilde{q}_i \in Q_i \backslash \{q_i\}) \quad \text{telle que} \quad V_i(\boldsymbol{b}_i, \tilde{q}_i) \geq V_i(\boldsymbol{b}_i, q_i) \tag{1.25}$$

Une politique qui n'est pas entièrement dominée est dite *utile*.

1.2.3.2 Résolution

La résolution d'un DEC-POMDP, c'est-à-dire la recherche d'une politique jointe optimale, constitue le cœur de notre travail, et nous allons la détailler plus profondément dans la partie suivante.

1.3 Résolution des Processus de Décision Markovien Distribués

L'apport principal de cette thèse consiste en la proposition d'un ensemble d'algorithmes pour la résolution, optimale ou approchée, des processus de décision markovien décentralisés. Nous allons présenter dans cette partie l'essentiel de ce travail, le situer dans son contexte scientifique et montrer les liens qui existent avec des travaux postérieurs en planification mono-agent, théorie des jeux, programmation dynamique et apprentissage par renforcement.

1.3.1 Résolution par Programmation Dynamique

La première partie est consacrée à la présentation de l'algorithme de programmation dynamique optimal introduit par Hansen et al., et de notre approche de programmation dynamique à base de points.

1.3.1.1 Programmation Dynamique pour DEC-POMDPs

L'algorithme de programmation dynamique présenté par Hansen et al. [HBZ04] a été le premier algorithme non-trivial permettant de résoudre de manière générale les DEC-POMDPs à horizon fini. Il consiste en une alternance de deux opérateurs : la génération exhaustive des politiques possibles, et l'élagage de politiques dominées.

Comme pour le cas d'un POMDP, un ensemble de politiques est évalué sur l'ensemble de l'espace des croyances. Ensuite, chaque politique qui est entièrement dominée par une combinaison d'autres politiques est supprimée. À la différence du cas mono-agent, chaque agent possède néanmoins son propre ensemble de politiques locales, et l'évaluation de ses politiques se fait dans l'espace des états de croyance multi-agents propre à cet agent. Le processus de génération, puis d'élagage de politiques est ensuite répété pour chaque horizon, en commençant par les politiques à horizon 1 (c'est-à-dire les actions simples). L'algorithme 7 résume l'approche de programmation

dynamique pour DEC-POMDPs. Le choix d'une politique jointe optimale pour un état initial s_0 se fait alors par simple maximisation sur les politiques utiles de l'horizon recherché :

$$\boldsymbol{q}^*_{s_0} = \underset{\boldsymbol{q} \in Q_1 \times \cdots \times Q_n}{\arg\max} V(s_0, \boldsymbol{q}) \tag{1.26}$$

L'approche de programmation dynamique proposée par Hansen et al. est aussi applicable à des systèmes compétitifs, puisque seuls les politiques entièrement dominées sont supprimées. Elle constitue ainsi un outil de résolution pour les jeux stochastiques partiellement observables (POSG). Malheureusement, cela signifie en même temps que l'approche n'est pas capable d'exploiter d'avantage l'aspect coopératif du problème, un point que nous allons aborder dans la partie suivante.

Algorithm 7 Programmtion Dynamique pour DEC-POMDPs - `ProgDynPourDEC-POMDPs()`

Require: Un DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$ et un horizon T
Ensure: Un ensemble de politiques utiles pour chaque agent

for all agent i **do**
  Initialiser $Q_i^0 \leftarrow \emptyset$
end for
for $t = 1$ **to** $t = T$ **do**
  /* Génération exhaustive de politiques */
  for all agent i **do**
    $Q_i^t \leftarrow$ `GenerationExhaustive`(Q_i^{t-1})
  end for
  /* Élagage itératif */
  repeat
    Trouver un agent i et une politique $q_i \in Q_i^t$ avec

$$(\forall \boldsymbol{b}_i)(\exists \tilde{q}_i \in Q_i^t \backslash \{q_i\}) \quad \text{tel que} \quad V_i(\boldsymbol{b}_i, \tilde{q}_i) \geq V_i(\boldsymbol{b}_i, q_i)$$

    $Q_i^t \leftarrow Q_i^t \setminus \{q_i\}$
  until l'élagage est complet
end for

1.3.1.2 Programmation Dynamique à Base de Points

L'opérateur de programmation dynamique à base de points aborde deux points faibles de l'approche de programmation dynamique présentée avant. D'une part, il est intéressant d'éviter la résolution coûteuse d'un programme linéaire, nécessaire pour l'identification des politiques dominées. D'une autre part, il est souvent bénéfique d'éviter de considérer des états de croyance qui sont peu probables ou simplement impossibles à atteindre.

Il a été montré pour la première fois par Lovejoy que la fonction de valeur d'un POMDP peut être approximée en discrétisant l'espace des croyances en une grille de régions [Lov91]. Au lieu de déterminer la fonction de valeur sur un espace continu, sa valeur n'est alors déterminée qu'aux seuls points définis par la grille : il suffit d'identifier les meilleures politiques aux points de la grille, ce qui veut dire que le programme linéaire est remplacé par un simple problème de maximisation discret. La valeur de tout état de croyance se trouvant entre les points définis par la grille peut alors être approximé par interpolation linéaire. La qualité de cette approximation dépend

évidemment de la résolution de la grille.

Pineau et al. ont proposé récemment une approche plus flexible, se basant sur des *points de croyance* qui ne sont plus distribués uniformément dans l'espace, mais aux endroits les plus intéressants pour le contrôle du système [PGT03]. Ces endroits peuvent être déterminés par des simulations de trajectoires markoviennes, c'est-à-dire par une génération stochastique de croyances possibles. Étant donné une croyance $\boldsymbol{b}$, une action a et une observation o, on peut générer une nouvelle croyance $\boldsymbol{b'}$ en utilisant la formule de Bayes

$$\boldsymbol{b'}(s') = \frac{\sum_{s\in\mathcal{S}} \boldsymbol{b}(s)\Big[\mathcal{T}(s,a,s')\mathcal{O}(s,a,o,s')\Big]}{P(o|\boldsymbol{b},a)}, \tag{1.27}$$

le dénominateur $P(o|\boldsymbol{b},a)$ étant le facteur de normalisation habituel. A partir de la croyance initiale $\boldsymbol{b_0}$, il est donc possible de générer des états de croyance pour tous les horizons du problème.

Notre travail est la synthèse de l'approche proposée par Pineau et al. et les problèmes de décision multi-agents. Nous montrons en effet qu'il est possible de générer un ensemble d'états de croyance multi-agent pertinents, ce qui constitue le cœur de notre algorithme de programmation dynamique à base de points. Contrairement au cas mono-agent, où la simulation de l'évolution du système suffit pour déterminer une distribution possible sur ses états, le cas multi-agent nécessite en plus la considération des distributions sur les politiques des agents. Nous différentions donc la croyance de l'agent i sur les états du système au moment t, que nous notons $\boldsymbol{b_i}^{\mathcal{S},t}$, et la croyance qu'il possède sur les politiques d'un autre agent j, que nous notons $\boldsymbol{b_i}^{j,t}$. Un état de croyance multi-agent complet pour un système à n agents se note donc $\boldsymbol{b_i}^t = \langle \boldsymbol{b_i}^{\mathcal{S},t}, \boldsymbol{b_i}^{1,t}, \ldots, \boldsymbol{b_i}^{i-1,t}, \boldsymbol{b_i}^{i+1,t}, \ldots, \boldsymbol{b_i}^{n,t}\rangle$.

En théorie des jeux, une distribution sur des politiques probables est souvent appelé *stratégie mixte*. On sait que l'utilité d'une politique (*stratégie* dans la nomenclature de la théorie des jeux) pour un agent i dépend en général de l'espace des stratégies mixtes de ses adversaires. Bien que nous nous plaçons dans un cadre coopératif et non compétitif, l'observabilité partielle, et donc l'incapacité de connaître avec sûreté l'état interne des autres agents, nous contraint de considérer plusieurs alternatives possibles en ce qui concerne leurs politiques locales - et donc des stratégies mixtes.

Comme nous l'avons décrit plus haut, la programmation dynamique pour DEC-POMDPs génère, à l'itération t, l'ensemble des politiques utiles Q_i^t pour les t derniers pas de l'exécution. Déterminer l'utilité d'une politique $q_i^t \in Q_i^t$ revient donc à répondre à la question suivante : quelles stratégies mixtes sur les politiques $\boldsymbol{Q}_{-i}^t$ des autres agents l'agent i doit-il considérer, sachant que $(T-t)$ actions jointes ont été exécutées et que $(T-t)$ observations jointes ont été perçues auparavant ? Nous allons montrer que chaque historique d'actions et d'observations jointes possibles nous mène effectivement à un ensemble d'états de croyance probables. Supposons que la politique jointe pour les $(T-t)$ premiers pas d'exécution ait été $\boldsymbol{q}^{T-t}$. Considérons maintenant un agent i, son point de vue sur le système, et plus précisément son information locale sur un autre agent j. Notons h_i^{T-t} l'*historique d'exécution* de l'agent i, c'est-à-dire l'historique des actions et des observations de longueur $(T-t)$. La connaissance de la politique jointe $\boldsymbol{q}^{T-t}$, des fonctions $\mathcal{T}$ et $\mathcal{O}$, et la formule de Bayes permettent alors de déterminer la probabilité $P(h_j^{T-t}|h_i^{T-t},\boldsymbol{q}^{T-t})$ de toute historique d'exécution h_j^{T-t} propre à l'agent j. L'agent j doit terminer son exécution avec une des politiques utiles $q_j^t \in Q_j^t$ pour les t derniers pas d'exécution. Ces politiques ont déjà

été déterminées par l'algorithme. Chaque historique d'exécution peut donc être associé à une de ces politiques, et la question qui reste est de savoir quelle politique associer à quelle historique. La probabilité d'une historique d'exécution revient donc à la probabilité d'une politique, et c'est ainsi que l'agent i peut déterminer les croyances possibles sur les politiques de l'agent j. En générant des historiques $\boldsymbol{h}_{-i}^{T-t}$ pour tous les agents sauf l'agent i, il est également possible de calculer une distribution sur les états sous-jacents du système, et c'est ainsi que l'agent i peut déterminer un ensemble d'états de croyance possibles à l'instant t.

En somme, tout agent i est donc capable de construire un ensemble d'états de croyance pertinents en (a) générant une politique jointe $\boldsymbol{q}^{T-t}$ pour les $(T-t)$ premiers pas de l'exécution, (b) choisissant une historique d'exécution locale h_i^{T-t}, (c) déterminant les probabilités des historiques $\boldsymbol{h}_{-i}^{T-t}$ des autres agents et (d) construisant un ensemble d'états de croyance consistant avec $\boldsymbol{q}^{T-t}$, h_i^{T-t} et les politiques possibles des autres agents $\boldsymbol{Q}_{-i}^{t}$. Cette approche est résumée dans l'algorithme 8. Nous appelons par la suite programmation dynamique à base de points *exacte*

Algorithm 8 Programmation Dynamique Multi-agent à Base de Points - `PDMABP()`

Require: Un DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$ et un horizon T
Ensure: Un ensemble de politiques utiles par agent

for all agent i **do**
  Initialiser $Q_i^0 \leftarrow \emptyset$
end for
for $t = 1$ **to** $t = T$ **do**
  for all agent i **do**
    for all politique jointe antérieure $\boldsymbol{q}^{T-t}$ **do**
      for all historique d'exécution h_i^{T-t} **do**
        $B(\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{\boldsymbol{Q}}_{-i}^{t}) \leftarrow$ Génération exhaustive des états de croyance
        for all $b_i^t \in B(\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{\boldsymbol{Q}}_{-i}^{t})$ **do**
          $Q_i^t \leftarrow Q_i^t \bigcup BR_i(b_i^t, \overline{Q}_i^t)$
        end for
      end for
    end for
    Remplacer $\overline{Q}_i^t \leftarrow Q_i^t$
  end for
end for

l'approche qui répète les parties de (a) à (d) pour toutes les configurations possibles. Comme elle ne considère que les états de croyance possibles, elle génère en général moins de politiques que l'approche de [HBZ04]. Néanmoins, elle doit considérer un nombre exponentiel d'états de croyance. Il est possible d'assouplir certaines conditions de l'approche exacte, ce qui mène à une version *approchée* de l'algorithme. Nous décrivons ici deux conditions qui peuvent être assouplies facilement.

Génération des politiques jointes Au lieu de considérer **toutes** les politiques jointes possibles, on peut adopter une stratégie déjà utilisée par [PGT03], à savoir la génération aléatoire d'un ou de plusieurs représentants. On peut utiliser la distance de Manhattan comme une métrique entre politiques, ce qui permet une répartition des politiques, en ne gardant que les politiques les plus éloignées les unes des autres.

Génération des états de croyance On peut choisir d'éviter la génération de **tous** les états de croyance. Chaque historique possède en général une probabilité d'apparition différente, et on peut établir une borne d'erreur ϵ sur le résultat final si on évite de considérer les historiques $\boldsymbol{h}_{-i}^{T-t}$ dont la probabilité d'apparition ne dépasse pas γ :

$$\epsilon \leq \frac{\gamma}{t(R_{max} - R_{min})} \tag{1.28}$$

1.3.2 Résolution par Recherche Heuristique

Nous allons développer dans ce paragraphe MAA*, un algorithme de recherche heuristique optimal pour la résolution des DEC-POMDPs. Il s'agit de l'extension multi-agent de l'algorithme de recherche heuristique A*. Nous allons d'abord caractériser l'espace de recherche considéré, poser par la suite la méthode d'évaluation et définir une classe de fonctions heuristiques admissibles. Nous démontrons enfin la complétude et l'optimalité de MAA*.

1.3.2.1 Problèmes à Horizon Fini

Nous avons vu qu'une politique optimale pour un POMDP à horizon fini T peut être représentée comme un arbre de décision q^T de profondeur T, donc un graphe de politique acyclique dans lequel les nœuds représentent les actions à effectuer, et où le parcours de l'arbre est choisi en fonction des observations reçues [KLC98]. De la même manière, les politiques optimales pour un ensemble de n agents peuvent être représentées sous forme d'une politique jointe $\boldsymbol{q}^T = \langle q_1^T, \dots q_n^T \rangle$. L'espace de recherche est donc donné par l'ensemble des politiques jointes $\boldsymbol{q}^t$ à horizons t, avec $0 \leq t \leq T$, où T est l'horizon du problème.

MAA* est l'extension multi-agent de l'algorithme classique A*. Il s'agit en effet d'une recherche multi-chemins, c'est-à-dire que n politiques optimales sont recherchées en parallèle. Comme pour A*, la recherche se fait de manière incrémentale : les feuilles de l'arbre de recherche, aussi appelées liste OUVERT et notées D [Pea90], contiennent des solutions partielles au problème, et à chaque itération, la meilleure solution partielle est choisie pour être développée d'une étape supplémentaire (*best-first*). Dans notre cas, une feuille de l'arbre contient donc une politique jointe à horizon $t < T$. Développer une politique jointe $\boldsymbol{q}^t$ à horizon t signifie construire tous les fils de $\boldsymbol{q}^t$, c'est-à-dire toutes les politiques jointes δ^{t+1} à horizon $(t+1)$ qui coïncident avec $\boldsymbol{q}^t$ pour les t premiers niveaux. Une partie d'un tel arbre de recherche est montrée en figure 1.4. Il a une profondeur de T et explore au pire cas l'espace de recherche tout entier, c'est-à-dire un nombre total de

$$\sum_{t=0}^{T} \left(|\mathcal{A}|^{\frac{1-|\Omega|^t}{1-|\Omega|}} \right)^n \tag{1.29}$$

politiques possibles, où T est l'horizon du problème, $\mathcal{A}$ et Ω correspondent aux ensembles des actions et des observations et n est le nombre d'agents. La base de tout algorithme de recherche heuristique est ensuite la décomposition de la fonction d'évaluation en une partie exacte pour une solution partiellement construite, et une estimation heuristique pour la partie restante. Dans notre cas, il s'agit donc d'évaluer des politiques jointes $\boldsymbol{q}^t$ pour un certain horizon t, et de trouver une heuristique H^{T-t} pour estimer le comportement potentiel du système après l'exécution de $\boldsymbol{q}^t$. Nous introduisons Δ^{T-t} comme le *complément* d'une politique jointe $\boldsymbol{q}^t$, c'est-à-dire un ensemble d'arbres de profondeur $(T-t)$ qui peuvent être attachés aux feuilles de $\boldsymbol{q}^t$, tel que $\langle \boldsymbol{q}^t, \Delta^{T-t} \rangle$ constitue une politique jointe complète de profondeur T. Les politiques $\boldsymbol{q}^t$ et $\langle \boldsymbol{q}^t, \Delta^{T-t} \rangle$ sont bien

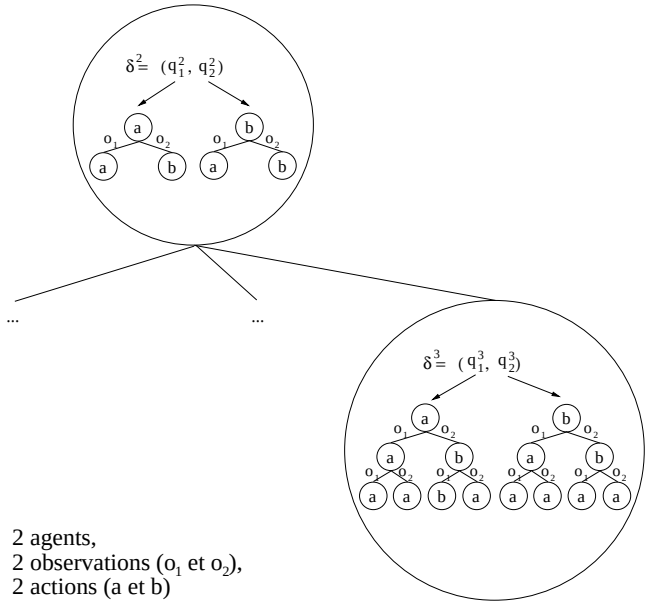

FIG. 1.4 – Une partie de l'arbre de recherche A* multi-agent. La figure montre une politique jointe à horizon 2 avec un de ses fils développés, une politique jointe à horizon 3.

définies et peuvent être évaluées en utilisant les paramètres du problème. Nous noterons $V(s_0, \boldsymbol{q}^t)$ et $V(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle)$ leurs évaluations. Nous définissons la différence entre ces deux évaluations comme la valeur du complément Δ^{T-t} :

$$V(\Delta^{T-t}|s_0, \boldsymbol{q}^t) = V(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle) - V(s_0, \boldsymbol{q}^t) \tag{1.30}$$

D'une manière équivalente, nous pouvons décomposer la valeur de toute politique jointe en la valeur d'une racine et la valeur du complément :

$$V(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle) = V(s_0, \boldsymbol{q}^t) + V(\Delta^{T-t}|s_0, \boldsymbol{q}^t) \tag{1.31}$$

Pour une politique jointe $\boldsymbol{q}^t$ partiellement construite, il est possible d'évaluer une borne supérieure sur son éventuelle valeur à horizon T en utilisant le meilleur complément possible :

$$\overline{V}^{*T}(s_0, \boldsymbol{q}^t) = V(s_0, \boldsymbol{q}^t) + \max_{\Delta^{T-t}} V(\Delta^{T-t}|s_0, \boldsymbol{q}^t) \tag{1.32}$$

Calculer cette valeur explicitement revient à une recherche exhaustive dans l'espace des politiques. La fonction heuristique H^{T-t} a pour but de *surestimer* efficacement la valeur du meilleur complément :

$$H^{T-t}(s_0, \boldsymbol{q}^t) \geq \max_{\Delta^{T-t}} V(\Delta^{T-t}|s_0, \boldsymbol{q}^t) \tag{1.33}$$

Une heuristique vérifiant cette propriété est dite *admissible* et, pour toute heuristique admissible, la fonction d'évaluation de l'algorithme A* multi-agent peut finalement s'écrire :

$$\overline{V}^{T}(s_0, \boldsymbol{q}^t) = V(s_0, \boldsymbol{q}^t) + H^{T-t}(s_0, \boldsymbol{q}^t) \geq \overline{V}^{*T}(s_0, \boldsymbol{q}^t) \tag{1.34}$$

La définition d'une heuristique admissible, c'est-à-dire d'une borne supérieure sur la valeur d'un DEC-POMDP partiel, est la partie essentielle de l'algorithme de recherche. Nous avons déjà

montré que la fonction de valeur d'un POMDP peut être facilement surestimée à l'aide du MDP sous-jacent. Nous allons montrer dans la suite qu'une propriété semblable peut être établie dans le cas multi-agent. Pour cela, nous posons $P(s|s_0, \boldsymbol{q})$ comme la probabilité que l'état du système soit s après l'exécution de la politique jointe $\boldsymbol{q}$ à partir de s_0. Nous définissons ensuite une deuxième heuristique h comme une borne supérieure de la valeur du DEC-POMDP optimal pour un horizon et un état initial bien défini

$$h^t(s) \geq V^{*t}(s) = \max_{\boldsymbol{q}^t} V(s, \boldsymbol{q}^t) \tag{1.35}$$

avec $h^0(s) = 0$. Ceci nous permet de définir toute une *famille* de fonctions heuristiques :

$$H^{T-t}(s_0, \boldsymbol{q}^t) = \sum_{s \in \mathcal{S}} P(s|s_0, \boldsymbol{q}^t) h^{T-t}(s) \tag{1.36}$$

Intuitivement, toute heuristique H est admissible, puisqu'elle correspond à ce qu'on obtiendrait en communiquant à chaque agent l'état global du système après l'exécution de la politique jointe $\boldsymbol{q}^t$. La connaissance de l'état constitue une information supplémentaire que les agents n'auront pas au moment de l'exécution. La valeur de H est donc supérieure à la valeur de toute politique. Nous insistons sur le fait que H est déjà une heuristique en soi, indépendamment de l'heuristique h, et H peut ainsi approximer une infinité de distributions alors que h ne doit être évaluée que pour les $|\mathcal{S}|$ états du système.

Lemma 1.3.1. *Toute fonction heuristique H est admissible, si h est admissible.*

L'intérêt principal de l'heuristique H est la simplicité de son évaluation, comparée au calcul de la fonction de valeur exacte. Dans notre cas, cette simplification a deux raisons. D'abord, la valeur de l'heuristique $h^t(s)$ peut être réutilisée dans le calcul de H dans l'équation (1.36) pour chaque nœud de l'arbre de recherche qui se trouve à la même profondeur t. Ceci réduit le nombre d'évaluations de $|\Omega^{t^n}|$ à $|\mathcal{S}|$. Ensuite, le calcul de h lui-même peut apporter des bénéfices en temps de calcul. Nous allons présenter dans la suite plusieurs façons de définir une fonction h admissible.

L'heuristique MDP Une première approche peut consister à considérer le MDP mono-agent sous-jacent, qui lui est complètement observable. Elle a déjà été employée par [Was96] puis par [GB98] pour résoudre des POMDPs par recherche heuristique. Pour un DEC-POMDP donné, on définit le MDP centralisé correspondant comme suit : $\mathcal{S}$ est le même ensemble d'états que dans le DEC-POMDP, $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ est l'ensemble des actions jointes, $\mathcal{P}(s, (a_1, \ldots, a_n), s')$ et $\mathcal{R}(s, (a_1, \ldots, a_n))$ sont identiques aux mêmes fonctions définies dans le DEC-POMDP. L'état courant du système est connu, il n'y a donc pas d'observations. Ceci implique que le contrôle du MDP peut être plus efficace que celui du DEC-POMDP, et sa fonction de valeur constitue par conséquent une borne supérieure de celle du DEC-POMDP correspondant :

$$h^{T-t}(s) = V^{T-t}_{MDP}(s) \tag{1.37}$$

Résoudre un MDP peut se faire à l'aide de méthodes de recherche ou par programmation dynamique, avec une complexité polynomiale au pire cas.

L'heuristique POMDP Une meilleure solution que celle que nous venions de considérer consiste à définir l'heuristique comme la valeur du POMDP centralisé sous-jacent. Il correspond au cas ou l'agent connaît à chaque instant les observations, et donc l'état interne, de tous les agents. La définition du POMDP sous-jacent inclut l'ensemble des observations jointes $\Omega = \Omega_1 \times \cdots \times \Omega_n$ et la fonction d'observation associée à la définition du MDP. Nous avons donc l'heuristique suivante :

$$h^{T-t}(s) = V^{T-t}_{POMDP}(s) \tag{1.38}$$

Une telle heuristique reste admissible, puisqu'elle simule toujours un système centralisé, bien que partiellement observable. Le POMDP sous-jacent permet de considérer les observations et les actions jointes, et donc de coordonner les agents d'une meilleure façon, ce qui n'est justement plus possible dans le cas de l'exécution décentralisée. Résoudre un POMDP est PSPACE-complet et nécessite le plus souvent une approche de programmation linéaire. Les variantes des algorithmes *witness* [KLC98] ou *incremental pruning* [CLZ97] peuvent être utilisées comme solution performante aux POMDPs. Toute borne supérieure au POMDP est évidemment elle-même admissible [Hau00]. Bien que l'heuristique POMDP soit plus complexe à calculer que l'heuristique MDP, le fait qu'elle soit plus proche de la vraie valeur du DEC-POMDP peut aboutir à une meilleure performance finale.

L'heuristique DEC-POMDP Un cas particulier de fonction heuristique consiste à utiliser la solution optimale au DEC-POMDP lui-même, calculée par exemple de manière récurrente par MAA* sur l'horizon restant :

$$h^{T-t}(s) = V^{T-t}_{DEC-POMDP}(s) = MAA^{*T-t}(s) \tag{1.39}$$

L'intérêt de cette heuristique s'explique par le fait suivant : en général, il existe un compromis entre la complexité de la fonction heuristique et sa proximité avec la vraie fonction de valeur. Ainsi, une fonction heuristique plus compliquée à calculer peut aboutir à un arbre de recherche moins large et par conséquent diminuer le temps de recherche global. La fonction h de l'équation (1.39) est admissible par définition, puisqu'elle constitue la vraie valeur du complément optimal si l'état initial était s. Les économies en temps de calcul viennent du fait que h ne doit être évalué qu'un nombre limité de fois, et que sa valeur peut être réutilisée dans les calculs de H dans (1.36) comme expliqué plus haut.

L'algorithme de recherche heuristique pour les DEC-POMDPs à horizon fini est présenté en Algorithme 9. MAA* est la synthèse d'une recherche dans l'espace des politiques jointes et de l'utilisation d'une des trois heuristiques (1.37), (1.38) et (1.39) dans le calcul de H. Comme le souligne le théorème suivant, cet algorithme est à la fois complet et optimal.

Theorem 1.3.1. *MAA* est complet et optimal.*

1.3.2.2 Problèmes à Horizon Infini

Tandis que, dans le cadre des processus de décision markoviens à horizon fini, le but est de trouver une politique optimale pour une durée de temps limitée, nous nous intéressons désormais à l'exécution de politiques sur un horizon infini. Cette problématique nécessite une autre approche. Nous allons donc développer dans ce qui suit l'extension de MAA* aux problèmes à durée infinie [SC05].

Algorithm 9 A* multi-agent - `MAA*()`

Require: Un DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$, un état initial s_0 et un horizon T
Ensure: Une politique jointe optimale

Initialiser la liste OPEN $D = \times_i \mathcal{A}_i$
$\boldsymbol{q}_{temp} \leftarrow \arg\max_{\boldsymbol{q} \in D} F^T(s_0, \boldsymbol{q})$
repeat
  /* Développer la meilleure solution courante */
  Choisir $\boldsymbol{q}^* \leftarrow \arg\max_{\boldsymbol{q} \in D} F^T(s_0, \boldsymbol{q})$
  $\boldsymbol{q}^{*'} \leftarrow$ `Développer(`$\boldsymbol{q}^*$`)`
  /* Tester si une nouvelle solution (sous-optimale) a été trouvée */
  if $F^T(s_0, \boldsymbol{q}_{temp}) < F^T(s_0, \boldsymbol{q}^{*'})$ **then**
    $\boldsymbol{q}_{temp} \leftarrow \boldsymbol{q}^{*'}$
    Afficher $\boldsymbol{q}^{*'}$
    for all $\boldsymbol{q} \in D$ **do**
      if $F^T(s_0, \boldsymbol{q}) \leq F^T(s_0, \boldsymbol{q}^{*'})$ **then**
        $D \leftarrow D \setminus \{\boldsymbol{q}\}$
      end if
    end for
  end if
  /* Ajouter une nouvelle politique jointe à la liste OPEN */
  $D \leftarrow D \bigcup \{\boldsymbol{q}^{*'}\}$
  if $\boldsymbol{q}^*$ est entièrement développée **then**
    $D \leftarrow D \setminus \{\boldsymbol{q}^*\}$
  end if
until $\exists \boldsymbol{q}^T \in D$ telle que $\forall \boldsymbol{q} \in D : F^T(s_0, \boldsymbol{q}) \leq F^T(s_0, \boldsymbol{q}^T) = V(s_0, \boldsymbol{q}^T)$

Contrairement au cas à horizon fini, trouver une politique optimale pour un POMDP à horizon infini n'est plus un problème décidable [MHC03], et ceci reste vrai pour les POMDPs multi-agent [BGIZ02]. La difficulté se traduit dans le fait qu'il est en général impossible de construire des arbres de politique à profondeur infinie. Résoudre un POMDP à horizon infini nécessite donc un choix préalable d'un modèle de politique. Nous allons nous limiter dans ce qui suit aux *automates finis déterministes*. Un automate fini déterministe pour un agent i est défini par un ensemble de nœuds $\mathcal{N}_i$ et deux fonctions, une fonction de sélection d'action $\psi_i : \mathcal{N}_i \rightarrow \mathcal{A}_i$ et une fonction de transition $\eta_i : \mathcal{N}_i \times \mathcal{A}_i \times \Omega_i \rightarrow \mathcal{N}_i$.

Comme pour le cas à horizon fini, la recherche s'effectue dans l'espace des politiques jointes partiellement construites. Un automate partiellement construit est un automate dont les fonctions de sélection d'action α et de transition η ne sont pas encore définies pour tous les nœuds. L'arbre de recherche va être initialisé avec un vecteur de *squelettes*, des automates "vides" d'une taille donnée mais où les fonctions α et η ne sont pas encore définies. Plus la recherche va progresser, plus la définition de ces fonctions va être complète. Le passage d'un squelette vers un automate entièrement défini est illustré par la figure 1.5. L'ordre du développement des nœuds peut être choisi librement, mais il est conseillé de définir α avant de définir η [MKKC99]. Si à chaque itération tous les automates sont développés simultanément, la profondeur de l'arbre de recherche va être de $|N| + |N||\Omega|$, puisqu'il y a $|N|$ actions et $|N||\Omega|$ nœuds successeurs à désigner. La figure 1.6 présente une partie du développement d'un tel arbre. La partie essentielle de

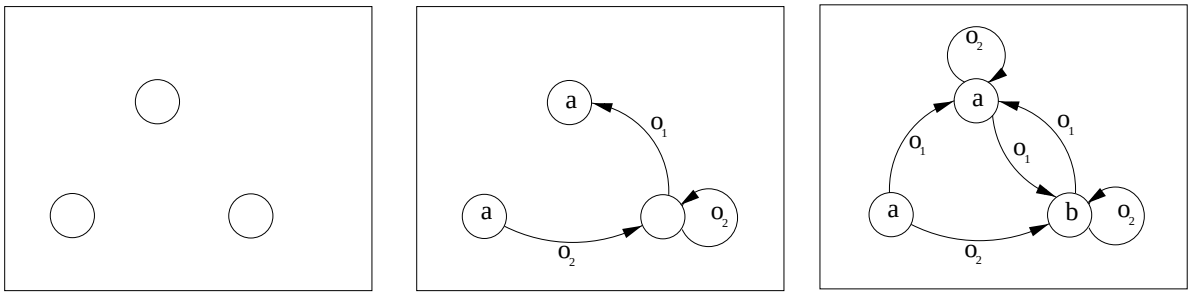

FIG. 1.5 – Les stades de développement d'un automate fini. A gauche, un squelette d'automate de taille 3, au centre l'automate partiellement développé, et à droite l'automate complètement développé (pour un environnement à 2 observations, o_1 et o_2).

la recherche incrémentale reste l'évaluation des politiques jointes partiellement définies. Pour garantir la propriété du *meilleur-d'abord*, l'évaluation heuristique de toute politique jointe doit être une borne supérieure de la valeur que l'on peut espérer si on complète cette politique de manière optimale. Contrairement au cas précédent, il faudrait donc trouver une borne supérieure pour un DEC-POMDP à horizon infini, ce qui n'est en général pas possible. Par contre, étant donné un DEC-POMDP et une politique jointe $\boldsymbol{q}$ partiellement définie, il est possible d'évaluer une borne supérieure sur la meilleure façon de compléter $\boldsymbol{q}$ pour le DEC-POMDP donné. Pour cela, nous allons définir deux fonctions supplémentaires, caractérisant respectivement pour chaque nœud si une action et un ensemble de nœuds successeurs ont déjà été choisis. $\Lambda_i(n)$ dénotera l'ensemble des actions possibles pour le nœud n de l'automate i, et $\Pi_i(n, o)$ dénotera l'ensemble des nœuds suivants possibles pour le nœud n et l'observation o de l'automate i :

$$\Lambda_i(n) = \begin{cases} \{\psi_i(n)\}, & \text{si } \psi_i \text{ est défini en } n \\ \mathcal{A}_i, & \text{sinon} \end{cases}$$

$$\Pi_i(n,o) = \begin{cases} \{\eta_i(n,o)\}, & \text{si } \eta_i \text{ est défini pour } n \text{ et } o \\ \mathcal{N}_i, & \text{sinon} \end{cases}$$

De la même manière, nous définissons les versions à n agents $\Lambda(\boldsymbol{n}) = \langle \Lambda_1(n), \ldots, \Lambda_n(n) \rangle$ et $\Pi(\boldsymbol{n}, \boldsymbol{o}) = \langle \Pi_1(n,o), \ldots, \Pi_n(n,o) \rangle$. Ceci nous permet de définir un nouveau MDP comme le produit vectoriel entre le DEC-POMDP et le vecteur de politiques. Il a déjà été établi pour le cas mono-agent que le produit vectoriel entre un automate fini et un POMDP constitue un nouveau MDP [Han97], [MKKC99]. Nous sommes capables d'établir un résultat équivalent dans le cas multi-agent :

Definition 1.3.1 (Produit vectoriel entre un DEC-POMDP et une politique jointe). *Soient un DEC-POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$ et une politique jointe partielle $\boldsymbol{q} = \langle \{\mathcal{N}_i\}, \{\psi_i\}, \{\eta_i\} \rangle$ donnés, nous définissons leur produit vectoriel comme un nouveau MDP $\langle \overline{\mathcal{S}}, \overline{\mathcal{A}}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \rangle$, avec*

- $\overline{\mathcal{S}} = (\underset{i}{\times} \mathcal{N}_i) \times \mathcal{S}$
- $\overline{\mathcal{A}} = \underset{i}{\times} (\mathcal{A}_i \times \mathcal{N}_i^{\Omega_i})$
- $\overline{\mathcal{T}}((s, \mathbf{n}), (\mathbf{a}, \eta^{\mathbf{n}}), (s', \mathbf{n}')) = P(s, \mathbf{a}, s') \sum_{\substack{\mathbf{o} \in \times \boldsymbol{\Omega} \\ s.t.\ \eta^{\mathbf{n}}(\mathbf{o}) = \mathbf{n}'}} O(s, \mathbf{a}, \mathbf{o}, s')$
- $\overline{\mathcal{R}}((s, \mathbf{n}), (\mathbf{a}, \eta^{\mathbf{n}})) = R(s, \mathbf{a})$

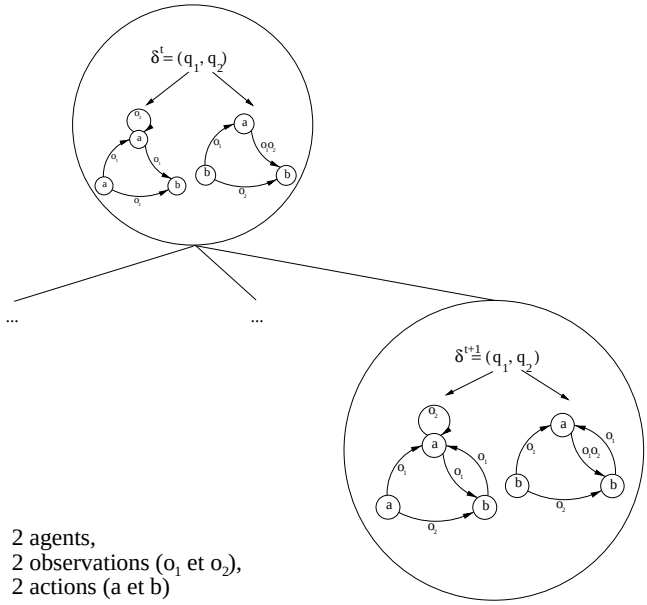

FIG. 1.6 – Une section de l'arbre de recherche pour le cas à horizon infini et un problème à 2 agents. Le nœud fils contient des automates qui sont complétés à un niveau de plus que ceux du nœud père.

et où $\eta^{\boldsymbol{n}}$ *défini une fonction qui - pour tout vecteur de nœuds* $\boldsymbol{n}$ *- retourne un vecteur de nœuds successeurs* $\boldsymbol{n'}$ *pour chaque vecteur d'observations* $\mathbf{o}$, $\eta^{\boldsymbol{n}} : \boldsymbol{\Omega} \rightarrow \mathcal{N}$.

Résoudre ce MDP peut se faire par des méthodes de programmation dynamique classiques, ce qui revient à trouver la fonction de valeur et le point fixe suivant :

$$\overline{V}(s, \boldsymbol{n}, \boldsymbol{q}) = \max_{\boldsymbol{a} \in \Lambda(\boldsymbol{n})} \left\{ R(s, \boldsymbol{a}) + \gamma \left[\sum_{s', \boldsymbol{o}} P(s', \boldsymbol{o} | s, \boldsymbol{a}) \max_{\boldsymbol{n'} \in \Pi(\boldsymbol{n}, \boldsymbol{o})} \overline{V}(s', \boldsymbol{n'}, \boldsymbol{q}) \right] \right\} \tag{1.40}$$

Cette fonction de valeur est l'extension multi-agent de la fonction donnée dans [MKKC99]. Si la politique jointe $\boldsymbol{q}$ est entièrement spécifiée, le MDP dégénère en une simple chaîne de Markov et la valeur $\overline{V}(s, \boldsymbol{n}, \boldsymbol{q})$ représente alors la valeur de la politique $\boldsymbol{q}$ en s si elle est exécutée à partir des nœuds $\boldsymbol{n}$. La valeur d'une politique jointe $\boldsymbol{q}$ est enfin :

$$\overline{V}(s_0, \boldsymbol{q}) = \max_{\mathbf{n} \in \boldsymbol{\mathcal{N}}} \overline{V}(s_0, \mathbf{n}, \boldsymbol{q}) \tag{1.41}$$

Si la politique $\boldsymbol{q}$ n'est pas encore entièrement spécifiée, alors $\overline{V}(s_0, \mathbf{n}, \boldsymbol{q})$ représente une surestimation de sa valeur.

L'extension de MAA* aux problèmes à horizon infini est présenté dans l'algorithme 10. Nous soulignons son optimalité, toujours en fonction de la classe de politiques considérée, par le théorème suivant :

Theorem 1.3.2. *L'algorithme A* multi-agent présenté dans l'algorithme 10 est complet et optimal pour le nombre de nœuds choisi.*

Algorithm 10 MAA* à horizon infini - `MAA*HorizonInfini()`

Require: Un DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$, un état initial s_0, et une taille de politique N
Ensure: Une politique jointe optimale

Initialiser la liste OPEN D avec le vecteur de squelettes de la taille N
$\boldsymbol{q}_{temp} \leftarrow \arg\max_{\boldsymbol{q} \in D} \overline{V}(s_0, \boldsymbol{q})$
repeat
  /* Développer la meilleure solution courante */
  Choisir $\boldsymbol{q}^* \leftarrow \arg\max_{\boldsymbol{q} \in D} \overline{V}(s_0, \boldsymbol{q})$
  $\boldsymbol{q}^{*'} \leftarrow$ `Développer(`$\boldsymbol{q}^*$`)`
  /* Tester si une nouvelle solution (sous-optimale) a été trouvée */
  if $\overline{V}(s_0, \boldsymbol{q}_{temp}) < \overline{V}(s_0, \boldsymbol{q}^{*'})$ **then**
    $\boldsymbol{q}_{temp} \leftarrow \boldsymbol{q}^{*'}$
    Afficher $\boldsymbol{q}^{*'}$
    for all $\boldsymbol{q} \in D$ **do**
      if $\overline{V}(s_0, \boldsymbol{q}) \leq \overline{V}(s_0, \boldsymbol{q}^{*'})$ **then**
        $D \leftarrow D \setminus \{\boldsymbol{q}\}$
      end if
    end for
  end if
  /* Ajouter une nouvelle politique jointe à la liste OPEN */
  $D \leftarrow D \bigcup \{\boldsymbol{q}^{*'}\}$
  if $\boldsymbol{q}^*$ is fully constrained **then**
    $D \leftarrow D \setminus \{\boldsymbol{q}^*\}$
  end if
until $\exists \boldsymbol{q}^* \in D$ entièrement définie telle que $\forall \boldsymbol{q} \in D : \overline{V}(s_0, \boldsymbol{q}) \leq \overline{V}(s_0, \boldsymbol{q}^*) = V(s_0, \boldsymbol{q}^*)$

1.3.3 Résolution par Apprentissage

Cette troisième partie ne concerne pas la planification mais l'apprentissage de politiques optimales pour un système multi-agent. Il s'agit là d'un problème plus compliqué que celui de la planification. Ceci peut être illustré en considérant l'exemple des deux agents et des deux ponts de la section 1.2.3.1. Nous avions constaté que la valeur d'une politique dans un environnement multi-agent dépend des politiques futures des autres agents. Déterminer une politique optimale pour un agent nécessite donc raisonner sur les politiques optimales éventuelles des agents restants. Ceci est possible si la planification est centralisée, c'est-à-dire si le planificateur peut contrôler le choix de politiques de tous les agents. Dans le cadre de l'apprentissage, qui lui est décentralisé, cela n'est plus possible : la complexité augmentée de l'apprentissage multi-agent vient donc de la nécessité d'une coordination supplémentaire lors du calcul des politiques. Dans un cadre plus général, Hart et Mas-Colell ont pu montrer que c'est précisément ce découplage qui complique la résolution décentralisée de tout problème multi-agent [HMC03].

Nous avons constaté que le modèle MDP et l'algorithme du Q-learning sont définis pour le cas où une seule action est choisie à chaque itération. Néanmoins, le formalisme peut être élargi à des problèmes où les actions peuvent être exécutées simultanément par plusieurs agents : il suffit de considérer toutes les actions jointes $\boldsymbol{a} = \langle a_1, a_2, \ldots, a_n \rangle$ comme actions atomiques et de redéfinir la fonction de transition et la fonction de récompense. Cette extension du forma-

lisme MDP s'appelle MMDP (MDP multi-agents) et a été proposée pour la première fois par Boutilier [Bou99]. Il est évident que tout MMDP peut ainsi être résolu de manière optimale en considérant le MDP avec actions jointes correspondantes. L'approche MMDP possède toutefois deux inconvénients. Il s'agit d'abord d'un apprentissage centralisé, c'est-à-dire que la tâche consistant à trouver une politique optimale pour le groupe ne peut pas être distribuée parmi les agents. En plus, la nécessité de considérer toutes les actions jointes augmente la complexité de l'apprentissage de façon considérable, puisque la taille de l'espace d'actions jointes croît de façon exponentielle en fonction du nombre d'agents.

Des agents qui prennent en considération les actions jointes, c'est-à-dire les actions de tous les autres agents, sont appelés *joint action learners*, tandis que les agents qui ne prennent en compte que leurs propres actions, et qui considèrent ainsi les autres agents comme faisant partie de l'environnement, sont nommés *independent learners* [CB98]. Nous allons nous intéresser dans la suite au deuxième cas. Notre but est de trouver un algorithme décentralisé avec communication pour des *independent learners*, qui est capable de trouver les mêmes solutions optimales qu'un apprentissage centralisé avec action jointes. Nous allons appeler une telle solution *MMDP-optimale*. Nous considérons la récompense globale du système comme étant la somme des récompenses individuelles, une propriété qui est appelée *indépendance des récompenses* [BZLG03]). Le système décentralisé est alors défini par :

- $\mathcal{S}$ l'ensemble des états du système, $s \in \mathcal{S}$,
- $\mathcal{A}_i$ l'ensemble des actions exécutables par l'agent i,
- $\mathcal{T}(s, \langle a_1, a_2, \ldots, a_n \rangle, s')$ la fonction de transition du système global,
- $r_i(s, a_i)$ les fonctions de récompense individuelles,
- $\pi_i(s)$ les politiques locales,
- $q_i(s, a_i)$ les Q-fonctions locales,
- $v_i(s) = q_i(s, \pi_i(s))$ les fonctions de valeur locales.

Nous allons décrire maintenant cet algorithme, appelé *apprentissage par notification réciproque* [SC04c, SC04a]. Pour cela, nous introduisons $\underline{q}_i(s, a_i)$ comme la valeur d'une *possible* mise à jour pour l'agent i. Elle est calculée comme pour le Q-learning dans des environnements déterministes :

$$\underline{q}_i(s, a_i) = r_i(s, a_i) + \gamma v_i(s') \tag{1.42}$$

Les actions des agents sont exécutées de manière synchrone et le système progresse de l'état s vers l'état s' suivant la fonction de transition $\mathcal{T}$. Chaque agent peut maintenant comparer la valeur de $\underline{q}_i(s, a_i)$ avec la valeur d'état de sa politique actuelle $v_i(s) = q_i(s, \pi_i(s))$, c'est-à-dire tester si la nouvelle valeur représente une amélioration locale ou pas :

$$\underline{q}_i(s, a_i) \stackrel{?}{>} v_i(s) \tag{1.43}$$

Le gain local pour l'agent i est alors

$$\Delta \underline{v}_i(s) = \underline{q}_i(s, a_i) - v_i(s) \tag{1.44}$$

et le gain global du système est

$$\Delta \underline{V}(s) = \sum_i \Big[\underline{q}_i(s, a_i) - v_i(s) \Big] \tag{1.45}$$

Il est évident que $(\Delta \underline{V}(s) > 0)$ *si* $(\exists i)(\Delta \underline{v}_i(s) > 0)$, c'est-à-dire que toute amélioration globale est perçue par au moins un agent au niveau local. Ainsi, dès que le test en (1.43) est positif pour

un agent i, celui-ci notifie tous les autres agents en leur communiquant la valeur $\Delta\underline{v}_i(s)$. Si une telle notification a eu lieu, chaque agent j répond alors en envoyant lui aussi sa valeur $\Delta\underline{v}_j(s)$, ce qui permet à tout agent de faire le calcul en (1.45) et de déterminer si une amélioration globale a eu lieu ou non. Dans le premier cas, un mise à jour est effectuée pour chaque agent :

$$\left\{\begin{array}{rcl} q_i(s,a_i) & \leftarrow & \underline{q}_i(s,a_i) \\ \pi_i(s) & \leftarrow & a_i \\ v_i(s) & \leftarrow & q_i(s,a_i) \end{array}\right. \tag{1.46}$$

Dans le deuxième cas, aucune mise à jour n'est effectuée. L'algorithme converge vers la solution optimale si la Q-fonction est initialisée avec des valeurs strictement inférieures à celles de la Q-fonction optimale, par exemple en posant $(\forall i, s, a_i)\ q_i(s,a_i) \leftarrow -\infty$. Il est résumé dans le schéma Algorithme 11.

Algorithm 11 Notification Réciproque - `NotificationReciproque()`

Require: Un MMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R} \rangle$ distribué
Ensure: Une politique jointe MMDP-optimale

while apprentissage **do**
  for all agents i **do**
    Exécuter une action a_i
    $\underline{q}_i(s,a_i) \leftarrow r_i(s,a_i) + \gamma v_i(s')$
    $\Delta\underline{v}_i(s) \leftarrow \underline{q}_i(s,a_i) - v_i(s)$
    if $\Delta\underline{v}_i(s) > 0$ **ou** des messages ont été envoyés **then**
      Diffuser $\Delta\underline{v}_i(s)$
    end if
    if communication a eu lieu **then**
      if $\sum_{j=1}^{n} \Delta\underline{v}_j(s) > 0$ **then**
        $q_i(s,a_i) \leftarrow \underline{q}_i(s,a_i)$
        $\pi_i(s) \leftarrow a_i$
        $v_i(s) \leftarrow q_i(s,a_i)$
      end if
    end if
  end for
end while

L'algorithme converge, sous les mêmes conditions que le Q-learning classique, vers la politique optimale, ce que nous soulignons par le théorème suivant :

Theorem 1.3.3. *L'algorithme de notification réciproque converge vers la solution MMDP-optimale du système.*

1.4 Expériences

Nous présentons ici brièvement les résultats de quelques expériences afin de valider la performance de nos algorithmes par rapport aux approches existantes, et de comparer les atouts et les inconvénients de ces approches par rapport aux problèmes choisis.

1.4.1 Problèmes Considérés

Le premier problème, introduit et spécifié dans [HBZ04], simule un réseau de communication simplifié. Les nœuds du réseau sont connectés entre eux, et pour chaque nœud i, une probabilité p_i d'arrivée d'un nouveau message est définie. Le message doit être diffusé à tous les nœuds voisins, mais une collision a lieu si deux nœuds essayent d'utiliser conjointement le même canal. Les messages ne sont alors pas envoyés et restent dans leurs tampons. L'objectif consiste à maximiser le nombre de messages transmis. Une récompense de +1 est donnée pour chaque message transmis. Nous avons testé notre algorithme sur un réseau de communication à deux nœuds, chaque nœud ayant un tampon capable de stocker un seul message. Il n'y a pas d'écrasement de messages dans un tampon, c'est-à-dire que les nouveaux messages ne peuvent arriver que si le tampon est vide. Le système peut ainsi être décrit par 4 états, à savoir les états des deux tampons (rempli/vide). Chaque nœud a 2 actions à sa disposition (envoyer/ne pas envoyer) et il y a 2 observations (collision/pas de collision). La transmission des messages et l'observation des collisions sont probabilistes, et les fréquences d'arrivée de nouveaux messages sont de $p_1 = 0.9$ et $p_2 = 0.1$.

Le deuxième problème simule une équipe de robots qui doivent se rencontrer dans un monde à deux dimensions et puis rester ensemble le plus longtemps possible. Les robots se déplacent sur une grille, mais la perception de chaque robot est limitée à sa propre position ; il n'a aucune information sur l'emplacement des autres robots. De plus, les actions des robots sont stochastiques et échouent dans 40% des cas, donc même si deux robots se sont rencontrés, ils peuvent à nouveau se perdre de vue, ce qui fait de ce problème un processus à horizon infini. Le problème que nous considérons est celui de [BHZ05] : il comporte 2 robots et une grille de 2×2 cases, ce qui représente donc 16 états, 5 actions par robot, ainsi que 4 observations, indiquant s'il y a un mur à droite et/où à gauche du robot. Une récompense de +1 est donnée à chaque fois que les robots se retrouvent sur la même case.

Le troisième problème est l'extension multi-agent du problème du tigre de Kaelbling, introduit par [NTY+03]. Il consiste en 2 agents qui se trouvent devant 2 portes fermées. Derrière l'une d'elles se trouve un tigre affamé, derrière l'autre en revanche se trouve une grande récompense. Chaque agent peut ouvrir une des deux portes de son choix, les agents peuvent ainsi décider d'ouvrir tous les deux la même porte ou chacun une porte différente. Chaque agent peut aussi écouter ce qui se passe derrière les portes, et obtenir ainsi une observation - bruitée - de l'état sous-jacent du système. La problématique consiste à déterminer comment les agents peuvent se coordonner pour d'abord obtenir une information suffisante à propos de la position du tigre, pour ensuite ouvrir ensemble la même porte. Dans la version A du problème, une haute récompense positive est donné lorsque la bonne porte est ouverte par les deux agents, mais une large pénalité est distribuée dès que l'un des deux agents ouvre la porte du tigre. Écouter devant une porte a un faible coût. Dans la version B du problème, les agents ne sont pas pénalisés s'ils ouvrent en même temps la porte avec le tigre.

Le quatrième problème simule un problème de foot. Sur une grille de dimensions 5×5, deux robots footballeurs doivent pousser une balle sur une case but. Chaque joueur peut choisir entre quatre actions de déplacement `Nord, Sud, Est, Ouest` et l'action de `Shoot`er la balle. Si le robot se déplace sur la case qui contient la balle, celle-ci est poussée sur la case suivante, tandis que l'action `Shoot` déplace la balle de deux cases. Une bonne stratégie de coordination consiste donc à utiliser l'action `Shoot` pour jouer une passe à l'autre robot.

1.4.2 Algorithmes de Planification

Nous avons d'abord comparés l'algorithme MAA* à horizon fini avec l'approche de programmation dynamique proposée par Hansen et al. [HBZ04]. La figure 1.7 montre le nombre total de paires de politiques que chaque algorithme doit évaluer, alors que la figure 1.8 compare la consommation d'espace mémoire des deux approches. Bien que l'approche de recherche heuristique ne considère pas nécessairement moins de politiques au total, ses besoins en espace mémoire sont beaucoup moins importants. Ceci implique que MAA* est capable de résoudre des problèmes où l'approche de Hansen et al. doit être abandonnée par manque d'espace mémoire. Ainsi, MAA* semble par exemple être le premier algorithme optimal qui puisse résoudre le problème du tigre multi-agent à horizon 4.

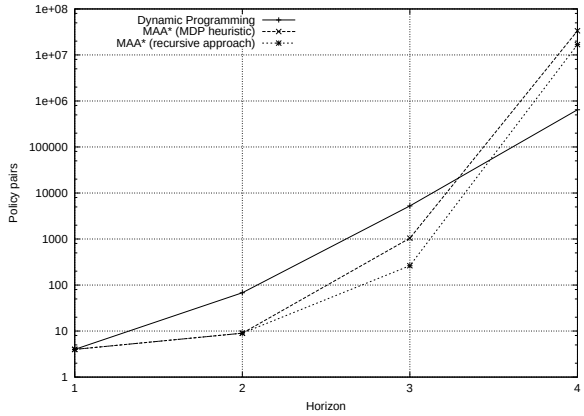

FIG. 1.7 – Le nombre de politiques évaluées pour l'approche de programmation dynamique, MAA* avec l'heuristique MDP et MAA* avec l'heuristique récursive pour le problème du réseau de communication (échelle logarithmique).

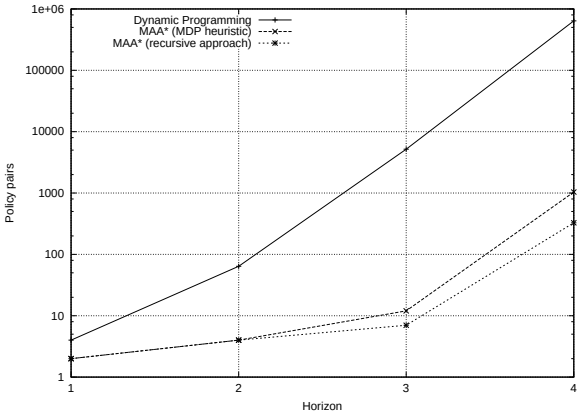

FIG. 1.8 – Les besoins en espace mémoire pour l'approche de programmation dynamique, MAA* avec l'heuristique MDP et MAA* avec l'heuristique récursive pour le problème du réseau de communication (échelle logarithmique).

Dans la Figure 1.9, nous avons comparé la recherche heuristique à horizon infini avec l'approche de *bounded policy iteration* (BPI) de [BHZ05], qui, à la différence de notre approche, optimise des contrôleurs stochastiques. On peut améliorer la performance de politiques probabilistes décentralisées en introduisant un contrôleur supplémentaire dit de *corrélation* [BHZ05]. Ce contrôleur est indépendant des autres, et le problème reste donc entièrement décentralisé, mais il permet une meilleure coordination entre les agents. La figure montre respectivement des résultats avec et sans contrôleur de corrélation. Tandis que notre approche est capable de trouver des solutions optimales pour des politiques déterministes, BPI est une approche d'optimisation locale dont les solutions peuvent représenter des optima locaux. Bien que d'un point de vue théorique, les automates stochastiques promettent une récompense moyenne plus élevée que les automates déterministes, il est apparemment plus difficile de les optimiser. En effet, les politiques déterministes trouvées par notre approche sont plus performantes que celles trouvées par BPI, que ce soit avec ou sans contrôleur de corrélation supplémentaire. Notons par contre que l'approche BPI permet de considérer des automates de plus grande taille.

Nous avons finalement implanté la version exacte et la version approchée de notre algorithme de

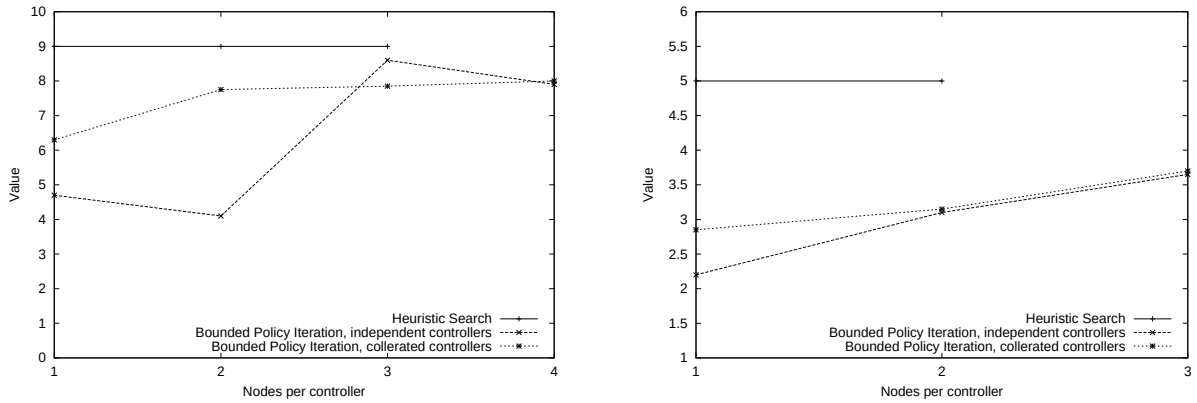

FIG. 1.9 – Valeur optimale des politiques déterministes pour l'approche de recherche heuristique, comparée à deux évaluations de *bounded policy iteration* pour des controlleurs stochastiques tirées de [BHZ05]. A gauche : Réseau de communication - A droite : Rencontre sur une grille.

programmation dynamique à base de points pour le problème simplifié du réseau de communication. Nous précisons que la version approchée que nous considérons consiste à générer une seule politique jointe, mais de considérer tous les fils d'exécution possibles. Les valeurs de la version approchée sont alors moyennées sur 10 essais.

L'obstacle majeur de l'algorithme de programmation dynamique multi-agent de [HBZ04] est l'explosion de sa consommation d'espace mémoire. Ainsi, l'algorithme ne peut pas résoudre le problème du réseau de communication pour des horizons plus grands que 4. La Figure 1.10 montre le nombre de politiques utiles, et donc la consommation de l'espace mémoire, identifiés par l'approche de programmation dynamique de Hansen et al. et nos approches à base de points (exacte et approchée). Alors que l'approche classique doit garder jusqu'à 300 politiques en mémoire, la considération explicité des états de croyance possibles réduit le nombre de politiques utiles considérablement. Nous sommes ainsi capables de traiter des problèmes plus larges.

Puisque nous évitons la résolution d'un programme linéaire, nécessaire pour supprimer les politiques dominées, et puisque nous considérons moins de politiques que l'approche de Hansen et al., le temps de calcul de notre approche est également réduit. La figure 1.11 résume les temps de calcul pour le problème du réseau de communication. Il est intéressant de constater que le temps de calcul connaît un pic pour l'horizon 5 pour redescendre après, mais notons que chaque horizon représente a priori un autre problème, et donc nécessite de considérer un autre ensemble de points de croyance. Notons aussi que l'échelle est logarithmique.

1.4.3 Algorithmes d'Apprentissage

Nous avons testé l'algorithme de notification réciproque sur le problème des robots footballeurs décrit en haut. La figure 1.12 montre le nombre de communications effectués pour différents seuils : Le cas sans seuil ($\delta = 0.0$) est comparé aux cas avec un seuil fixe ($\delta = 1.0, 2.0, 5.0$) et finalement à deux cas avec un seuil variable (seuil linéaire et décroissant de 10.0 à 2.0 ainsi que exponentiel et décroissant de 10.0 à 2.0). La convergence de la fonction de valeur est montrée dans la figure 1.13. Il est intéressant de remarquer que l'introduction d'un faible seuil ($\delta = 1.0$) baisse

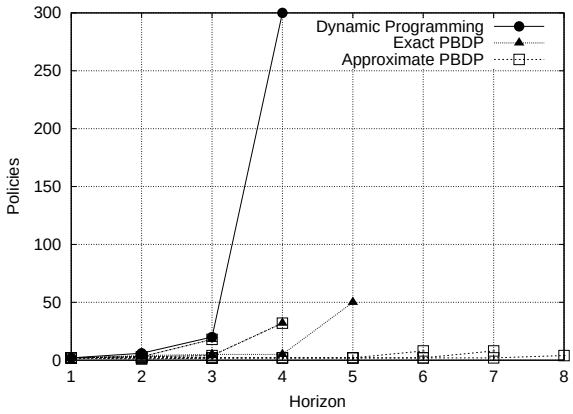

FIG. 1.10 – Le nombre de politiques utiles pour le problème du réseau de communication, évaluées par l'algorithme de Hansen *et al.*, la version exacte et finalement la version approchée de l'approche de programmation dynamique à base de points (PDBP).

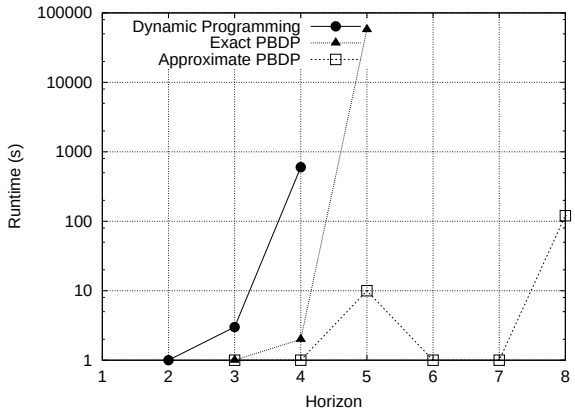

FIG. 1.11 – Le temps de calcul pour les trois approches de programmation dynamique multi-agent sur le problème du réseau de communication.

le nombre de communications par un facteur trois sans néanmoins ralentir le taux de convergence. Les seuils variables peuvent encore diviser par deux le nombre de communications effectuées : Comparé au pire cas *communication à chaque itération*, on peut obtenir une diminution du nombre des communications jusqu'à un facteur de 40 pour l'exemple traité.

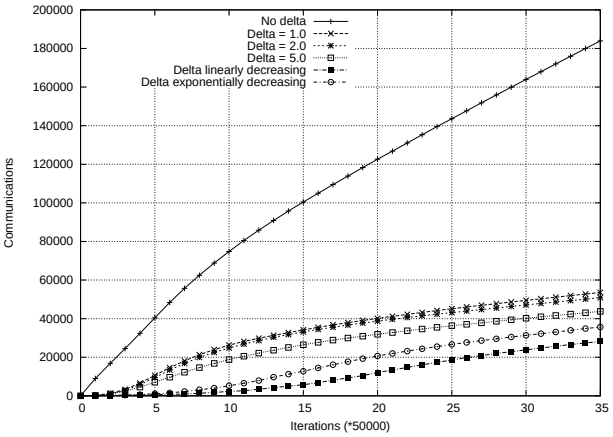

FIG. 1.12 – Le nombre total de communications effectuées pour 3 seuils fixes différents, deux seuils variables et le cas sans seuil.

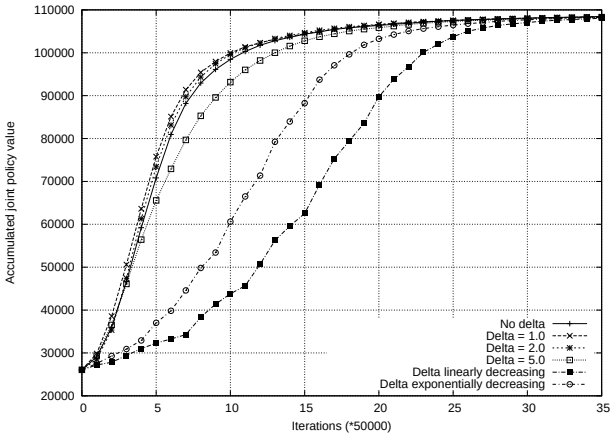

FIG. 1.13 – La valeur de la politique jointe des deux robots sur tous les états et pour tous les seuils utilisés.

1.5 Discussions

Nous avons traité et résolu dans cette thèse le problème du contrôle optimal décentralisé, formalisé sous forme d'un modèle particulier des Processus de Décision Markoviens, les DEC-POMDPs. Nous avons proposé les deux premiers algorithmes optimaux de recherche heuristique qui per-

mettent la résolution des DEC-POMDPs à horizon fini, ainsi que des DEC-POMDPs à horizon infini, lorsque les politiques sont représentées sous forme d'automates déterministes. Nous avons introduit une nouvelle version de programmation dynamique à base de points multi-agent, et nous avons finalement adressé le problème de l'apprentissage par renforcement multi-agent. Nous avons démontré l'optimalité de nos algorithmes, et nous avons pu illustrer, à travers des exemples applicatifs, les avantages que possèdent nos algorithmes vis-à-vis des algorithmes existants.

Dans la version complète et anglophone de la thèse, nous avons par ailleurs étudié certaines propriétés des systèmes multi-agent sous forme de DEC-POMDPs. Nous avons identifiés des structures particulières qui peuvent faciliter la résolution du DEC-POMDP général, nous avons introduit plusieurs concepts dans la théorie générale des DEC-POMDPs tels que la valeur de l'information contenue dans un acte de communication ou la version multi-agent du problème du *credit assignment*, et nous avons apporté quelques remarques critiques concernant la recherche actuelle dans le domaine.

En raison de la complexité au pire cas inhérente aux DEC-POMDPs, il est néanmoins inévitable de se focaliser dorénavant sur des approches approximatives, moins exigeantes en temps de calcul et en consommation de mémoire. D'un côté, il a été montré que même le calcul d'une politique jointe ϵ-proche de la politique optimale ne change en rien la complexité du problème [RGR02]. D'un autre côté, on peut espérer qu'un grand nombre de problèmes admettent une solution plus facilement calculable.

Chapitre 2

Introduction

2.1 Informal Introduction

We address in this thesis the problem of letting a family of computers decide together about the optimal progression of a dynamic system. We dedicate this introduction to familiarize the reader with the topic.

One can assume that human beings are faced with millions of decisions to make every day. Some of them may be obvious, some of them may be the result of years of reasoning. A decision problem covers the process of selecting the right action based on all information that is available. The action then entails a possible evolution of the environment in which the decision maker is situated, and eventually makes available new information. This closes a very simple sequential loop. Since we want a computer to make decisions, and since todays computers are discrete machines, we focus our attention solely on discrete systems : the sets of available actions as well as possible pieces of information are finite. The game of chess is a good example of such a discrete and sequential decision making problem.

Making a real decision means having a preference over several alternatives. This preference can be motivated in different ways. A human being for example would eat what he likes most, a man would marry to the prettiest girl in town [1]. If we translate the notion of preference into the language of computers, we usually assign values to the possible alternatives. A higher value could stand for a delicious meal, whereas a lower value could mean that the girl is rather ugly. The association between preferences and real numbers is very old, and has numerous origins. In psychology, Edward Thorndike defined what he called the "Law of Effect", stating that animals learn to prefer actions based on whether their consequences are mostly good or bad. In learning theory, these evaluations are often called rewards, hence the primary goal in computer-based decision making : accumulate the maximum amount of rewards. The aim of planning is determining a sequential policy of acting that achieves this goal.

Selecting an action that maximizes immediate rewards can be straightforward, especially if the parameters of the environment are fully known. Sometimes however, the reward can be delayed, such as in the game of chess for example, where it can only be determined at the end. In other cases, an interesting immediate reward may lead to a configuration of the environment that is

[1] One of my reviewers noted that this statement could be interpreted as being sexist, so I may have to write *heterosexual* man.

very disadvantageous in the long run. The delay of rewards is one reason why taking optimal decisions can be complicated. Another reason is stochasticity : in most cases, the outcome of an action is not deterministic, which is why a distribution over possible outcomes has to be considered. Making decisions under uncertainty while taking into account all of its possible long-term effects constitutes one of the major challenges in automated planning.

As research in robotics and artificial intelligence advances, the issue of collaborative decision making becomes more and more eminent. Instead of asking how to program a single artificial entity to allow for decision making capabilities, the question is how to program a set of such entities in such a way as to allow them to decide *together*. As one would expect, this includes the issues of communication, negotiation, or reasoning about the preferences of the other decision makers. Many approaches can be - and have been - imagined to address these subjects, such as inventing artificial languages, or establishing automated voting protocols. However, few work has been done in establishing a truly formal theory of optimal computer-based collaborative decision making that captures all of these issues.

We will present in the following the theory of decentralized control, on which our work is based on. This includes a mathematical framework to which, we believe, all aspects of decentralized decision making can be reduced to.

2.2 Decentralized control theory

One of the most widely used frameworks for automated planning, that captures the above mentioned properties of sequentiality, discreteness, and stochasticity, is the Markov decision process (MDP), named after the Russian mathematician Andrei Andreyevich Markov (1856-1922). The MDP model was mainly introduced by Richard Bellman and Ron Howard [Bel57, How60] and includes a formal definition of the environment, its dynamics under the influence of actions, and the generation of appropriate reward values. In an MDP, the dynamic system is considered to be in one of several possible states at each time step. The execution of an action will lead to a, possibly stochastic, state transition. At the same time, a real-valued reward signal is generated. The MDP model is at the same time very simple and very general, which is the reason why MDPs have been proven to be useful in a wide range of areas. It has even been extended to deal with continuous problems, asynchronous settings, and many more.

Unfortunately, the underlying system state, on which the action selection is based on, can not always be accessed directly. Consider for example a medical application, where the decision problem consists in determining the appropriate treatment for a patient. It is usually not straightforward to determine the exact nature of the disease the patient is suffering from. Nevertheless, the physician has to develop a well-founded preference over the possible treatments to undertake, based on partial and uncertain information. The MDP model has been extended to account for these cases, leading to the partially observable Markov decision process (POMDP) [Dra62, Ast65, Son71]. In a POMDP, the decision making process includes Bayesian reasoning, namely the adjustment of the *a posteriori* information state based on new evidence. Solving POMDPs is significantly harder than solving MDPs, because it basically translates to solving a continuous problem, but some major advances have been made to compute these solutions more efficiently. A brief historical introduction to stochastic control theory can be found in [Jr.96].

Suppose now a system in which the course of action depends on multiple decision makers, each one with its own perception of the situation. A difficult surgery for example could require the simultaneous participation of several doctors, which have to harmonize in such a way as to succeed in the operation. Such a setting usually includes problems of coordination, communication, and reasoning about the decisions taken by all other persons involved. In a computer-based system, this translates to having multiple agents with different local information about the system. Sharing the information between agents in order to coordinate best may be costly, and it may even not always be necessary, which is why it has to be taken into account when determining optimal control strategies. The problem of determining which local information is necessary to be communicated for the global operation to succeed, and which information can be deducted without distribution makes multi-agent problems significantly harder than their single-agent counterparts, and while the MDP and POMDP decision models have been established some time ago, their natural extension to multiple concurrent decision makers has been addressed only recently, leading to the decentralized POMDP model (DEC-POMDP) [BZI00, BGIZ02].

The research effort that is presented in this thesis can be summarized as analyzing the problem of decentralized control, and determining centralized planning algorithms for solving distributed stochastic decision problems.

2.3 Scientific Context

In the following section, we will establish a brief link between our work and connected research areas as well as previous efforts to deal with related problems, and we will provide some historical background to the problem of decentralized decision making in general.

2.3.1 Artificial Intelligence

Artificial intelligence is the science of simulating an intelligent and human-like behavior by an artificial entity. It has been established as a research area of its own at the Dartmouth conference in 1956, where John McCarthy finally coined the term *artificial intelligence*, and covers a wide range of disciplines, such as formal logic, neural networks, language processing, computer vision, machine learning, and many others. Historical milestones in AI include the definition of artificial neurons, so called *perceptrons*, whose behavior tends to imitate the functionality of biological neurons [MP43, Ros58, MP69], the establishment of a formal test, the *Turing test*, to determine whether a machine is capable of performing human-like conversations [Tur59], the creation of the artificial therapist ELIZA by Joseph Weizenbaum [Wei65], the development of the medical expert system MYCIN by Ed Feigenbaum, Bruce Buchanan, Edward Shortliffe and others, or the recent advances in robotic soccer of the RoboCup league. A good overview of research in AI can be found in [RN95].

2.3.2 Game Theory

Game theory has been established as a tool for modeling and analyzing economic behaviors at the beginning of the 20th century, mainly due to the works of Antoine Augustin Cournot [Cou38], and John von Neumann [vNM44]. It deals with the interactions between competing agents. A game is a well defined mathematical object, defined by a set of *players*, a set of *strategies* available to each player, and a *payoff matrix*, which assigns a value to each player for a given combination of strategies. Game theory can for example be used to analyze the behavior of shareholders at

the stock markets. One of its main research interests consists in determining possible optimality criteria. Solving a game can be seen as finding a set of strategies that satisfies the largest number of players. This however is often a difficult undertaking, and the existence of a solution that promises optimal payoffs for all participating agents can in general not be guaranteed. It is known however that competitive systems tend to converge to stable configurations, which are referred to as *equilibria*. John Nash proved that each game is guaranteed to exhibit at least one such equilibrium [Nas51].

Game theory covers a more general case of stochastic distributed systems than the one addressed in this thesis, since the payoff for each agent may in general be different. In game theory, agents thus usually follow different - and often conflicting - objectives, whereas we focus our attention on cooperative systems with a common payoff function for all agents. However, game theory provides some important insides into the evaluation of multi-agent strategies in general, and some of the concepts that have been developed in game theory are indispensable when solving multi-agent decision problems.

2.3.3 Multi-agent Systems

A multi-agent system is commonly referred to as a system in which intelligent agents pursuit a set of tasks by interacting with each other and with the environment in which they are situated. An agent is an autonomous entity that is able to perceive its environment through sensors, and act on it through effectors. Agents are usually supposed to be rational, which means that they maximize a given performance measure. Multi-agent systems are fully decentralized, i.e. there is no central authority that guides the agents. Examples of such agents are autonomous vehicles, robots, or software bots. Exhaustive summaries of the research done in the multi-agent systems community can for example be found in the works of Gerhard Weiss [Wei99] and Michael Wooldridge [Woo01].

Agents may be very simple stimulus-response entities, but they may also be more complex, including sophisticated reasoning capabilities. The field of *swarm intelligence* for example deals with the emergence of complex behaviors exhibited by extremely simple agents [ESK01], whereas the soccer playing agents of the RoboCup league are driven by very complex on-board vision and path planning algorithms. We are particular interested in *reactive agents*, which means agents that do not possess a representation of their environment and are thus not able to do any complex on-line reasoning about the decision to make. The decentralized control problems we consider in our work are mathematically well defined reactive multi-agent systems.

2.3.4 Reinforcement Learning

Reinforcement learning is an attempt to formalize the insights obtained by analyzing the psychology of animal learning, and to apply it to computer-based control problems. It is heavily based on Thorndike's "Law of Effects" [Tho11] and the notion of trial-and-error learning. It was Marvin Minsky who discussed the possibility of establishing a computational model of trial-and-error learning in his PhD thesis [Min54]. A reinforcement learner is an agent that establishes an evaluation function over all situations it might possibly be faced with, and who is then able to modify this function through continuous interactions with its environment. Claude Shannon suggested that such an idea could be used to program a chess playing machine [Sha50], and Arthur Samuel constructed his checkers player on a similar idea [Sam59].

Systems that learn based on reinforcements have led to impressive results. Rémi Coulom for example developed an artificial swimmer, based on work in motor control and artificial neural networks, that learned to articulate its members in such a way as to advance in a fluid [Cou02]. The reinforcement learning community has contributed to control theory by promoting the notion of artificial rewards and the concept of value functions. It also addresses the fundamental *credit assignment problem*, namely the question of which reward to assign to which action [Min61]. As Sutton and Barto explain it in their exemplary textbook about reinforcement learning : "How do you distribute credit for success among the many decisions that may have been involved in producing it ?" [SB98]. If we could assign accurate reward values for each move in a chess game, then playing chess would become an obvious undertaking. We do in general not address the problem of credit assignment in our work, but we assume that a reward function is given.

2.4 Applications

The problem of decentralized decision-making occurs in numerous real-world scenarios, and before presenting algorithms to solve it, we depict some example problems where our algorithms can be applied to.

2.4.1 Multi-rover exploration

In January 2004, the two NASA rovers *Spirit* and *Opportunity* landed on the surface of Mars with the goal to search for and characterize a wide range of rocks and soils, which might give some hints about the possible presence of water on Mars. The mission *Mars Exploration Rovers* was commonly said to be a success, and in March 2004, the NASA presented photos made by *Opportunity* that prove the earlier existence of water on Mars. The communication with the rovers occurs through the two unmanned spacecrafts *Mars Odyssey* and *Mars Global Surveyor* that are orbiting Mars, but the communication time is very precious, and the rovers usually act on their own due to on-board path-planning and obstacle-avoidance software.

The mission *Mars Exploration Rovers* is a typical example of a decentralized decision problem, where the rovers can be thought of as agents, navigating in a stochastic environment that they perceive only partially through noisy sensors. The goal is to explore the surface in a manner that is most efficient, considering the robot's power consumption, the relevance of the photos and samples taken, and the data that is finally sent back to Earth. Such a problem can be formalized as a distributed Markov decision process [ZWBM02, BZLG04].

2.4.2 Rescue Robots

Rescue robots are being developed to access devastated areas during or after large scale disasters such as fires, earthquakes, or military attacks, that are not accessible by humans. Their goals usually include extinguishing fire places, locating injured persons, or clearing ruined areas. In order to do so, the robots have to be able to locate themselves, and to plan collaborative rescue missions that may require the participation of several specialized robots. The robots have to act under uncertainty, and they usually have to discover and explore the environment at the same time while operating. Together with the rapidly changing environment and the pressure of time, this constitutes a major challenge for intelligent multi-agent systems [KT01], and has been established as the primary research goal of the *RoboCupRescue* league. An example scenario is

FIG. 2.1 – A photo of the Mars rover *Spirit*, taken by the rover itself on February 21st 2004. The source of the photo is http ://photojournal.jpl.nasa.gov/catalog/PIA05339.

presented in Figure 2.2. The robot rescue problem can be thought of as a dynamic optimization problem, where robots have to maximize a cumulated reward over time. The reward may for example translate the number of people rescued, or the amount of fire extinguished. The problem can therefore be formulated as a decentralized Markov decision problem.

2.4.3 Communication Networks

Decentralized decision making also occurs in numerous kinds of computer networks, such as the Internet. A common issue is packet routing, namely the decision which packet to send to which node in order to maximize the networks throughput, or to minimize the average transfer time per message. It is known that there exists no global view of a system like the Internet, and the decision problem the nodes have to face is thus inherently decentralized. Each node of the network constitutes an agent, which has only a partial view of the entire system. These agents, often called *routers*, have to estimate the reliability and the speed of its outgoing channels based on past observations. Costs are incurred when packets are lost or delayed in queues. Nodes of the network may communicate with each other in order to acquire information about the local state of the network, but it is important to emphasize that these communication acts themselves constitute the parameters that should be optimized. The theory of Markov decision processes has been applied to modeling computer networks, as shown by Altman [Alt00], and an attempt to solve the routing problem using reinforcement learning has been suggested by Peshkin and Savova [PS02].

2.4.4 Nanorobotics

As proclaimed by Richard Feynman in 1959, "There's Plenty of Room at the Bottom". The research efforts undertaken in the field of the *very small* are nowadays known under the name of *nanoscience*. A recent but very promising subfield of nanoscience is that of *nanorobotics*. Nanorobots are extremely simple entities, usually of the size of a few molecules, that can perform same basic actions. The world's first single-molecule nanocar is shown in Figure 2.3 and described in

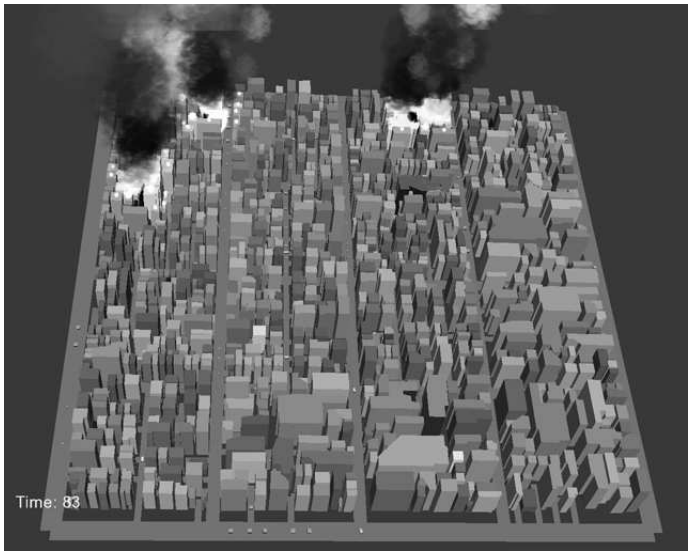

FIG. 2.2 – The simulation of a burning city in the RoboCupRescue league. The source of the photo is http ://kaspar.informatik.uni-freiburg.de/∼rescue3D/pics/shot_vc_burn.jpg.

[SOZ+05].

Many possible applications of nanorobotics can be imagined, for example nanosurgey. Such a problem indeed perfectly fits our framework : the nanorobots are simple reactive agents with a small action set, the observability of the environment is limited, and there is in general no communication possible between the nanomolecules. The robots have to accomplish a cooperative task under uncertainty, the output of which might be critical for the well-being of the patient. It is thus crucial to establish an optimal strategy, which is guaranteed not to fail under any circumstances. The field of nanorobotics is still at its beginnings, but we believe that the application of

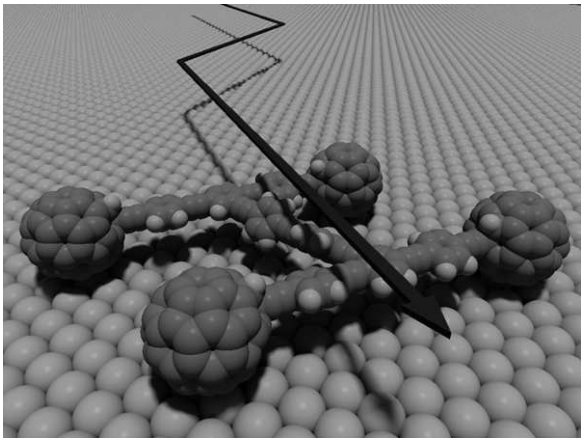

FIG. 2.3 – The world's first single-molecule nanocar, developed by Kevin Kelly and James M. Tour at Rice University. The source of the photo is http ://www.rice.edu/media/NanoCartriangle.tif.

the algorithms presented in this thesis might be of great value.

2.4.5 Sensor Networks

A sensor network is a network of many, spatially distributed devices that use sensors to monitor conditions at different locations, such as temperature, sound, pressure, pollution, and many more. The sensors communicate their local information to a monitoring computer. Sensing consumes resources, and since the devices are usually light and have small batteries, it is important to determine smart sensing strategies, such that the information gain is maximized while keeping the power consumption low. One issue in sensor networks is for example that of avoiding the gathering of redundant information. Solving decision problems related to sensor networks has recently been done using the Markov decision process framework [NVTY05].

2.5 Contributions

The contributions of this thesis are the following.

We introduce :
- an extension of heuristic search to distributed decision problems,
- the definition of MAA* (multi-agent A*) and appropriate admissible heuristic functions,
- an infinite-horizon version of MAA*, based on finite-state controllers,
- the first algorithmic approach to compute relevant multi-agent belief states,
- a multi-agent point-based dynamic programming, synthesis of the single-agent point-based approximation and multi-agent dynamic programming,
- an approximated version of point-based multi-agent dynamic programming,
- the mutual notification algorithm, an optimal reinforcement learning algorithm for multi-agent systems.

We also define or identify :
- the multi-agent credit assignment problem,
- the value of information associated to a message exchange in multi-agent MDPs,
- symmetric and uniform DEC-POMDPs,
- the need for individual reward functions in the case of cooperative multi-agent reinforcement learning.

2.6 Outline

The aim of this thesis is threefold. We first give an overview of the mathematical models for optimal centralized and decentralized decision making, and recent algorithms that can be used to solve these models. We then contribute by introducing several new algorithms to solve decentralized control problems. We are finally confident to have identified throughout the manuscript some new interesting problem classes, challenging future research problems, as well as critics concerning some current approaches. The remainder of this thesis is thus organized as follows.

Chapter 2 introduces the mathematical framework used throughout our work, namely the Markov decision problem. We define the MDP and the POMDP model, we describe what is commonly understood as solving these problems optimally, and we give a brief introduction into basic planning techniques, using dynamic programming and heuristic search.

Chapter 3 describes decentralized extensions of the Markov decision process, above all the DEC-

POMDP which we will be using as our basic formalism. We briefly mention alternative models for decentralized decision making, we cover communication between agents, and we particularly mention different subclasses of the general DEC-POMDP that exhibit particular structure.

Chapter 4 covers game theoretic algorithms for solving DEC-POMDP. Although game theory is a more general field, and although algorithms coming from game theory do not necessarily guarantee optimality in the DEC-POMDP case, their insight is very important in order to understand the subsequent algorithms for solving DEC-POMDPs optimally.

Chapter 5 covers dynamic programming algorithms for solving DEC-POMDP. In this chapter, we introduce the basic dynamic programming algorithm, proposed by Bernstein et al., and our decentralized point-based dynamic programming algorithm. This includes a detailed analysis of the concepts of multi-agent belief states, dominance of strategies, and pruning. The ideas of this chapter were published in [SC06].

Chapter 6 covers heuristic search algorithms for solving DEC-POMDP. This is were we will introduce multi-agent A* (MAA*), the extension of heuristic search techniques to decentralized decision problems, both over finite horizons [SCZ05] and infinite horizons [SC05]. It has been shown that MDP value functions can be used to establish upper bounds for POMDPs, and we extend this result to specify a class of admissible heuristic functions for DEC-POMDP search algorithms.

Chapter 7 covers solving DEC-POMDPs by reinforcement learning. Learning decentralized policies is a more challenging problem than planning for them, and we show how some of the classical reinforcement learning algorithms can be applied to multi-agent problems. In particular, we introduce the mutual notification algorithm, published in [SC04c].

Chapter 8 contains the description of some experimental setups and performance results of our algorithms in comparison to other approaches from the literature.

Chapter 9 summarizes our contributions, points out some strengths and possible weaknesses, and draws some lines for possible directions of future research.

We believe that Chapter 2 and Chapter 3 should be read first in order to understand the mathematical foundations of decentralized Markov decision problems, while Chapters 4, 5, 6, and 7 can be studied mostly independently of each other.

Chapitre 3

Markov Decision Processes

3.1 Markov Decision Processes

This chapter is dedicated to the introduction of the mathematical decision theoretic formalism that enables to solve single-agent control problems. The theory of sequential decision making has particularly been pushed forward by Richard Bellman [Bel57] and Ronald Howard [How60] in the early 60s. Their research finally led to the definition of a formal framework, the *Markov decision process* (MDP). MDPs are discrete descriptions of dynamic systems that enable not only to simulate but also to qualify the evolution of the system. They are strongly related to decision theory and the problem of optimal control. Basic MDPs make some strong assumptions about the decision process. There cannot be more than one single decision making entity, the agent, which in addition has to have full access to the current state of the system. The MDP formalism has nevertheless been found to be very useful in a variety of areas. An exhaustive introduction to Markov decision processes can be found in the textbooks by Puterman [Put94] and Bertsekas and Tsitsiklis [BT05, BT01].

3.1.1 The MDP Formalism

In the following, we introduce the general MDP model of sequential decision making on which all our work is based on.

Definition 3.1.1 (MDP). *A* Markov decision process *MDP is defined as a tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$*, where*

- $\mathcal{S}$ *is a finite set of states* $s \in \mathcal{S}$*,*
- $\mathcal{A}$ *is a finite set of actions* $a \in \mathcal{A}$*,*
- $\mathcal{T}(s, a, s')$ *is the probability of transitioning from state* s *to state* s' *after execution of action* a*,*
- $\mathcal{R}(s, a)$ *is the reward produced by the system when action* a *is executed in state* s*.*

There exist some variants of the formalism, especially concerning the reward function. Sometimes rewards are associated to a complete state transition $\mathcal{R}(s, a, s')$, sometimes they are just defined for states only $\mathcal{R}(s)$. These variants however are mostly equivalent to each other.

Executing an MDP means repeating the following steps over the horizon of the problem : observing the current state of the system, choosing an action, observing the state transition, and collecting the associated reward. The sequence of states the system traverses is commonly called a *Markov chain*.

3.1.1.1 The Markov Property

The fundamental assumption of the MDP formalism is that it verifies the *Markov property*. This property states that the probability of transitioning from state s to state s' is independent from the past :

$$P(s_{t+1} = s' | s_0, a_0, s_1, a_1, \ldots, s_t, a_t) = P(s_{t+1} = s' | s_t, a_t) = \mathcal{T}(s_t, a_t, s_{t+1}) \tag{3.1}$$

The Markov property ensures that the current state is always sufficient to predict the behavior of the system and to make optimal decisions. A large number of real-world processes verify the Markov property, and even non-markovian processes can usually be transformed into markovian ones by adapting the definition of the state space to include histories of states. One can formulate this principle the other way around : the state space of an MDP should be defined in such a way as to allow making optimal decisions based on the current state only.

3.1.1.2 Performance Criteria

A performance criterion aggregates the single-step rewards produced after each state transition and assign them a single value at the end of the process. This value can be interpreted as being the usefulness of the associated Markov chain. Several performance criteria can be established, and we briefly present the two criteria we will be using for our algorithms.

The Finite-horizon Criterion For a finite-horizon Markov chain, the simplest aggregation method consists in summing all individual rewards into one value :

$$\sum_{t=0}^{T-1} \mathcal{R}(s_t, a_t) \tag{3.2}$$

The Discounted Infinite-horizon Criterion For Markov chains of infinite length, the unbound accumulation of rewards poses a problem, which is why an additional discount factor is usually introduced to keep the evaluation finite :

$$\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \qquad \text{with} \qquad 0 \leq \gamma < 1 \tag{3.3}$$

Establishing a performance criterion leads to the following question : which action sequence does an agent have to chose in order to maximize the criterion ? The answer to this question is usually not obvious, partly because of the stochasticity of the environment and the possible delay of rewards : the same state-action pair may usually lead to several different successor states, and transitions that produce low immediate rewards may lead to successor states with much higher expectations afterwards. The problem the decision maker is faced with consists in controlling the system in such a way as to produce the most useful Markov chains under the selected performance criterion. A rule that enables making such decisions is called a *policy*. It is usually denoted π.

Definition 3.1.2 (Markov Decision Problem). *A* Markov Decision Problem *is an MDP with an associated performance criterion. Solving a Markov decision problem means finding a policy π that, if it is being executed within the MDP, optimizes the performance criterion.*

Given a policy π and a performance criterion, an associated *state value function* V can be established in order to define the expected utility of that policy for any given state. For finite-horizon problems, this value function can be defined as :

$$V(s,\pi) = E\left[\sum_{t=0}^{T-1} \mathcal{R}\left(s_t, a_t\right) \;\middle|\; s_0 = s\right] \tag{3.4}$$

For infinite-horizon problems and the discounted performance criterion, the value function becomes :

$$V(s,\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}\left(s_t, a_t\right) \;\middle|\; s_0 = s\right] \tag{3.5}$$

Given the definition of a value function, we can state the formal definition of the Markov decision problem as finding a policy that maximizes the value function for a given start state s_0. Any such a policy will be called an *optimal policy*, and will be denoted π^* :

$$\pi^* = \arg\max_{\pi} V(s_0, \pi) \tag{3.6}$$

In the most general case, the policy at time step t will be a function that maps the entire history of states and actions until t to a new action :

$$\pi_t(s_0, a_0, s_1, a_1, \ldots, s_t) \rightarrow \mathcal{A} \tag{3.7}$$

This means that a total number of $\mathcal{A}^{|\mathcal{S}|^t|\mathcal{A}|^{t-1}}$ policies are candidates for the optimal decision function at time step t. Because of the Markov property however, most of these policies are not needed, since it is sufficient to consider only those policies that depend on the current state :

$$\pi_t(s_t) \rightarrow \mathcal{A} \tag{3.8}$$

It is obvious that this dramatically reduces the size of the candidate set. It has furthermore been shown that the set of policies can be restricted to deterministic functions.

An obvious, albeit not very efficient way of obtaining an optimal policy would consist in exhaustively searching through the whole policy space, and in evaluating separately each possible resulting Markov chain. Richard Bellman made the important discovery that we can do much better than this :

Definition 3.1.3 (Bellman's Principle of Optimality [Bel57]). *"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."*

Bellman's optimality principle stipulates that any optimal policy can be computed incrementally, horizon by horizon. The technique of incrementally computing policies and value functions is usually called *dynamic programming*.

3.1.1.3 Factored MDPs

The above definition of the MDP model is based on a very simple state description. It is indeed sufficient that states can be enumerated in same very basic manner. In various domains however, states are correlated in some way or another. If for example the position of a robot is captured by a state, then there is a notion of proximity that can be exploited during policy computation :

states that are close to each other will tend to lead to similar outcomes under the effect of the same action. The original MDP model does not allow to express such properties as proximity or similarity.

Factored MDPs constitute an alternative description for MDPs. In a factored MDP, the state is not an atomic entity, but consists of several *factors* or features. These factors can be given *a priori*, or they can be learned [DSW06]. The state transition function, and later the policy, can then exploit this structure [BDG95]. If we return to the robot example, then we realize that the state space of the robot is inherently factored if Cartesian coordinates are being used : there are two factors, the x-coordinate and the y-coordinate, which constitute a state $s = (x, y)$. We do not go into the details of factored MDPs, but they often enable a much more compact representation of the MDP's dynamics, using Bayesian networks for example [Gue03].

3.1.2 Planning Algorithms

Optimal planning for Markov decision problems is the process of building an optimal policy while having the full knowledge about the dynamics of the system, namely its transition and reward functions. Planning is thus an off-line process. At runtime, the decision making agent only executes its plan, while it has no knowledge about how good or bad it actually scores. We will see later that things are different in the learning case. The main cycle of MDP policy execution is shown in Figure 3.1.

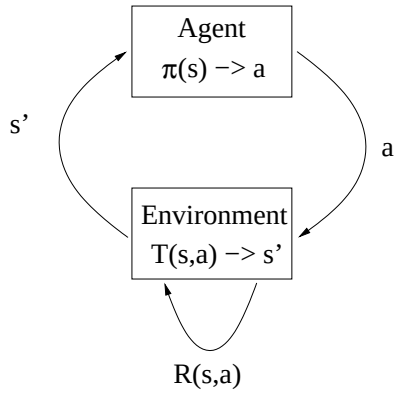

FIG. 3.1 – The basic cycle of policy execution : an agent choses an action based on its internal policy π, the environment transitions to a new state, and a reward signal $R(s, a)$ is produced and stored.

3.1.2.1 Policy Iteration

The first step towards solving an MDP optimally consists in being able to evaluate any given policy. Suppose a finite-horizon problem, and a policy $\pi = (\pi_T, \pi_{T-1}, \ldots, \pi_1)$, such that $\pi_t(s)$ denotes the action that should be executed in state s with t steps to go. Evaluating policy π can be done by unrolling its value function, thus applying the Bellman principle :

$$V(s, \pi_t) = \mathcal{R}(s, \pi_t(s)) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi_t(s), s') V(s', \pi_{t-1}) \tag{3.9}$$

Since the evaluation of the horizon t value function requires knowledge about the horizon $(t-1)$ value function, a typical algorithm for evaluating a finite-horizon policy starts by evaluating the last horizon first, and then by backpropagating these values to the previous horizons. A sketch of such an approach is shown in Algorithm 12. Evaluating an arbitrary policy is a necessary

Algorithm 12 Finite horizon policy evaluation - `PolicyEvaluation()`

Require: A MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ and a policy π_T
Ensure: The value function V for policy π_T
1: $V(s, \pi_0) \leftarrow 0, \quad (\forall s \in \mathcal{S})$
2: **for** $t = 1$ **to** $t = T$ **do**
3: **for all** $s \in \mathcal{S}$ **do**
4: $V(s, \pi_t) \leftarrow \mathcal{R}(s, \pi_t(s)) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi_t(s), s') V(s', \pi_{t-1})$
5: **end for**
6: **end for**

condition for improving it. One way of achieving a policy improvement consists in searching, for each state, if there is not a different action choice that could lead to a higher reward. This can easily be done by considering the expected value of a different immediate action, and following the existing policy thereafter, leading to a new function, the *state-action value function* Q :

$$Q(s, a, \pi_t) = \mathcal{R}(s, a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') V(s', \pi_{t-1}) \tag{3.10}$$

The state-action value function is directly related to the state value function as follows :

$$V(s, \pi) = \max_{a \in \mathcal{A}} Q(s, a, \pi) \tag{3.11}$$

For a given policy π and its value function V, it is thus possible to compute an improved policy π' through one-step lookahead of the state-action value function :

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q(s, a, \pi) \tag{3.12}$$

This is the basic step of *policy improvement*, which is shown in Algorithm 13. It can be proven that

Algorithm 13 Finite horizon policy improvement - `PolicyImprovement()`

Require: A MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ and a horizon T, a policy π and its value function V
Ensure: An improved policy π'
1: **for all** $s \in \mathcal{S}$ **do**
2: $\pi'(s) \leftarrow \mathcal{R}(s, a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') V(s', \pi_{t-1})$
3: **end for**

the policy π', obtained from the policy improvement step, is at least as good as the original policy $\pi : V(s, \pi') \geq V(s, \pi), \quad (\forall s \in \mathcal{S})$. Repeated cycles of policy evaluation and policy improvement lead to a globally optimal policy and constitute the heart of *policy iteration*, which is shown in Algorithm 14.

Algorithm 14 Finite horizon policy iteration - `PolicyIteration()`

Require: A MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ and a horizon T
Ensure: An optimal policy π_T^* for horizon T
1: Initialize a policy π^0 at random
2: $n \leftarrow 0$
3: **repeat**
4: $V \leftarrow$ `PolicyEvaluation(`π^n`)`
5: $\pi^{n+1} \leftarrow$ `PolicyImprovement(`π^n, V`)`
6: $n \leftarrow n + 1$
7: **until** $\pi^n = \pi^{n-1}$

3.1.2.2 Value Iteration

We have seen that solving the optimality equation (3.6) can be done by evaluating and improving policies. The main drawback of the previous policy iteration algorithm is that the policy evaluation step has to be executed over the entire horizon, although the policy improvement step only alters the first action. The Bellman principle however leads to the following observation : an optimal horizon T policy $\pi^* = (\pi_T^*, \pi_{T-1}^*, \ldots, \pi_1^*)$ also contains an optimal horizon $(T-1)$ policy, namely the policy $(\pi_{T-1}^*, \ldots, \pi_1^*)$. More generally, it contains optimal policies for all horizons $t \leq T$. For the computation of an optimal horizon t policy, it is thus sufficient to know an optimal horizon $(t-1)$ policy, and to add a one-step policy at its head. Computing such a one-step policy means finding the best action for each state, which means optimizing a mapping over a total of $|\mathcal{A}|^{|\mathcal{S}|}$ possibilities. Repeating this step T times leads to a total number of $T \cdot |\mathcal{A}|^{|\mathcal{S}|}$ policies. This is the complexity of the dynamic programming approach for computing an optimal finite-horizon policy. It is called *backward induction*, and it is given in Algorithm 15.

The concept of backward induction can also be applied to solve infinite-horizon problems. It has indeed been shown that optimal policies for infinite-horizon MDPs are stationary [Put94], which means that the decision functions for all horizons are identical $\pi_1^* = \pi_2^* = \cdots = \pi_\infty^* = \pi^*$. The optimal decision for a given state is thus time independent. This also implies the existence

Algorithm 15 Backward induction - `BackwardInduction()`

Require: A MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ and a horizon T
Ensure: An optimal policy π_T^* for horizon T
1: $V(s, \pi_0) \leftarrow 0, \quad (\forall s \in \mathcal{S})$
2: **for** $t = 1$ **to** $t = T$ **do**
3: **for all** $s \in \mathcal{S}$ **do**
4: $\pi_t(s) \leftarrow \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') V(s', \pi_{t-1}) \right]$
5: $V(s, \pi_t) \leftarrow \mathcal{R}(s, \pi_t(s)) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi_t(s), s') V(s', \pi_{t-1})$
6: **end for**
7: **end for**

of a unique optimal value function V^* for the problem. Determining this value function can theoretically be achieved by infinite backward induction. An approximation of the optimal value function can be obtained by iterating until an appropriate error bound is met. Infinite back-

ward induction is called *value iteration* and is shown in Algorithm 16. When value iteration

Algorithm 16 Value iteration - `ValueIteration()`

Require: A MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ and an error bound ϵ
Ensure: An ϵ-approximation of the optimal value function
1: $V^0(s) \leftarrow 0, \quad (\forall s \in \mathcal{S})$
2: $n \leftarrow 0$
3: **repeat**
4: **for all** $s \in \mathcal{S}$ **do**
5: $V^{n+1}(s) \leftarrow \max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \gamma V^n(s') \right]$
6: **end for**
7: $n \leftarrow n + 1$
8: **until** $||V^n - V^{n-1}|| \leq \epsilon$

has converged, an optimal policy can easily be extracted from any value function by one-step lookahead :

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \gamma V^*(s') \right] \tag{3.13}$$

The value iteration algorithm can be interpreted as the repeated application of an operator to the value function. This operator is often called the *Bellman operator* H, and it is defined as follows :

$$H(V)(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \gamma V(s') \right] \tag{3.14}$$

It has been shown that the Bellman operator is a contraction mapping, and that the optimal value function V^* is its unique fixed point :

$$V^* = HV^* \tag{3.15}$$

The fixed point property can also be written in its extensive form :

$$V^*(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \gamma V^*(s') \right] \tag{3.16}$$

It is commonly called the *Bellman optimality equation.*

3.1.2.3 Linear Programming

There exist other planning methods for solving MDPs optimally. The Bellman equations can for example be represented as a system of linear equations. By realizing that the Bellman operator is *monotonic*

$$V \leq HV \leq H^2V \leq \ldots \leq H^kV \tag{3.17}$$

and that the optimal state value function constitutes its unique fixed point

$$\lim_{k \to \infty} H^k V = V^* \tag{3.18}$$

one can state that $V \leq V^*$. The optimal state value function thus is the highest function that verifies the system of constraints $V \leq HV$. It can be obtained by solving the linear program

$$\begin{array}{rl} \text{maximize} & \sum_{s\in\mathcal{S}} \alpha_s V(s) \\ \text{constraint to} & V(s) \leq \mathcal{R}(s,a) + \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s')\gamma V(s'), \qquad (\forall s \in \mathcal{S})(\forall a \in \mathcal{A}) \end{array}$$

FIG. 3.2 – Linear program for solving MDPs.

given in Figure 3.1.2.3, with, in general, $\sum \alpha_s = 1$. Although solving an MDP through linear programming has longtime been considered as being a hard task compared to iterative approaches such as value iteration [Put94], it is being used more frequently for approximation techniques [Gue03, dFvR03]. For more details, the reader is referred to the textbooks by Puterman [Put94] or Bertsekas and Tsitsiklis [BT01].

3.1.3 Reinforcement Learning

Contrary to the previous sections, where we discussed off-line planning techniques given a model of the environment, learning a policy means discovering the transition model $\mathcal{T}$ *while* interacting with the environment. The main learning cycle of reinforcement is as follows : the agent chooses an action, a state transition occurs, and the agent observes a new state. Together with the new state, it receives a reward that gives an indication about the usefulness of the transition with regard to the problem to solve. The reward signal is then used to update a state-action value function, and a new action is chosen. Such a cycle is shown in Figure 3.3. The goal of

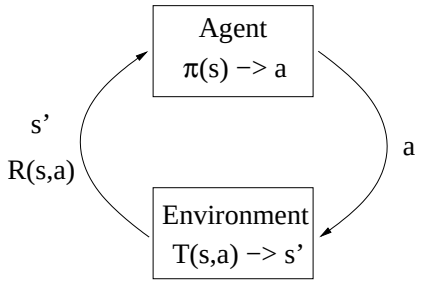

FIG. 3.3 – The basic cycle of reinforcement learning : an agent choses an action based on its internal policy π, the environment transitions to a new state, and provides the agent with an associated reinforcement signal $R(s,a)$.

the algorithm designer is to determine an update function of the state-values that enables the extraction of a near-optimal policy π in the long-run, which means a policy that maximizes the global amount of reward collected by the agent when following its action choices. Learning problems are usually infinite-horizon problems, although some finite-horizon variants do exist. An exhaustive introduction to reinforcement learning can be found in the textbook by Sutton and Barto [SB98].

3.1.3.1 Q-learning

One of the most popular reinforcement learning algorithms is Q-learning, introduced by Watkins [Wat89, WD92]. Q-learning is based on the on-line approximation of the state-action value function, where $Q(s, a)$ represents the expected accumulated reward when executing action a in state s, and following an optimal policy thereafter. Watkins showed how to approximate the state-action value function during the interaction with the system : each time an action a is executed in state s, and the system transitions to a new state s', the corresponding state-action value is updated using the following rule :

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a')\right] \tag{3.19}$$

The update simulates the compromise of keeping the old Q-value, which summarizes the history of the process, and replacing it with the current one-step lookahead of the value function. The compromise is guided by a weight variable $\alpha \leq 1$, which is usually decreased during learning in order to stabilize the Q-function. Q-learning is the on-line version of the value iteration algorithm for settings where the transition function itself is unknown. Watkins could show that the Q-values converge to the optimal state-action values when the interaction with the environment is long enough, and when all state-action pairs are selected sufficiently often [WD92].

Convergence For the general Q-learning algorithm to converge, two assumptions about the infinite sampling and the decaying of the learning rate have to be satisfied.

Assumption 3.1.1. *All state-action pairs (s, a) with $s \in \mathcal{S}$ and $a \in \mathcal{A}$ have to be visited infinitely often.*

Assumption 3.1.2. *The learning rate α satisfies :*

- $0 \leq \alpha$
- $\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} (\alpha_t)^2 < \infty$

Several authors then showed that Q-learning, and several other methods of value-function based reinforcement learning, converge to the optimal value function as long as they guarantee some standard contraction properties [WD92, SL99, JJS94].

Exploration Strategies A commonly used exploration strategy for Q-learning choses each action with a probability proportional to its state-action value. This guarantees that all actions will be chosen sooner or later, while assuring that more promising actions will be chosen more frequently :

$$P(a|s) = \frac{Q(s,a)}{\sum_{a' \in \mathcal{A}} Q(s,a')} \tag{3.20}$$

An extension of this strategy is the Boltzman strategy, which includes an additional temperature parameter K that can be decreased during learning :

$$P(a|s,k) = \frac{e^{\frac{Q(s,a)}{K}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s,a')}{K}}} \tag{3.21}$$

The Boltzman strategy allows the decision maker to guide the exploration/exploitation tradeoff. By reducing the temperature parameter, the agent exploits more, and a temperature of 0 is finally equivalent to a deterministic policy.

Watkins' Q-learning is finally given in Algorithm 17. There exist extensions of the basic Q-

Algorithm 17 Q-learning - `QLearning()`

Require: A MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$
Ensure: An approximation of the optimal Q-function
1: $Q(s,a) \leftarrow 0, \quad (\forall s \in \mathcal{S})(\forall a \in \mathcal{A})$
2: **loop**
3: Execute action a according to exploration strategy
4: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a') \right]$
5: **end loop**

learning algorithm that integrate explicit predictions about the behavior of the environment based on rules, such as *learning classifier systems*. Especially for factored MDPs, this often leads to quicker and better generalization properties : if a particular action has a very restricted influence on only a single factor of the current state, then its effect can be predicted using a simple *if-then* rule. Consider a grid world, where the factored state consists in the position of a robot (x, y). The effect of the action `North` for example could then be described as :

$$\texttt{North}(\cdot, y) \ \rightarrow \ (\cdot, y-1)$$

The point $\cdot$ indicates that the first factor of the state description remains unchanged whatever its value is. The efficiency of applying Q-learning methods to learning classifier systems has been investigated by Gérard et al. [GMS03].

3.1.3.2 Issues in Reinforcement Learning

There are several issues related particularly to reinforcement learning that do not necessarily appear in the case of planning. In the following, we will briefly mention some of them, without going into details.

The Model Problem In a reinforcement learning scenario, the underlying model of the system, e.g. its transition function $\mathcal{T}$, is unknown to the agent, otherwise it could apply planning techniques to optimize its behavior. The question is then whether it is more beneficial to first learn the model of the environment, and then plan an optimal control strategy based on this approximation, or if it is smarter to directly learn a state-value function while interacting with the environment. Most reinforcement learning approaches follow the second idea.

The Exploration/Exploitation Dilemma Learning to act optimally in an unknown environment means exploring its entire state-action space, since no guarantee can be given that an unknown state-action pair might not be the source of a very high reward. At the same time, visiting state-action pairs that a very costly can decrease the performance of the overall learning process. The exploration/exploitation dilemma translates the following problem : if an agent has already learned a partially optimal policy, should it rather continue following this policy in order

to gain rewards, or should it deviate from the policy, hoping that unknown state-action pairs might lead to even higher rewards. A very simple example for this dilemma is the multi-armed bandit problem : if a decision maker faces a slot machine with several levers, is it better to continue pulling on one lever, or should another lever be tried ? This is particularly interesting if the problem has a fixed predetermined horizon. There are several exploration strategies that allow to address this problem, and we will mention two of them in the next section.

The Credit Assignment Problem The definition of an appropriate reward function for solving a particular problem is what is known as the *credit assignment problem* (see also Section 4.6.2). It is particularly interesting in the case of reinforcement learning, since it is the immediate reward signal that guides the agent during learning. The credit assignment problem is the machine learning variant of a problem that biologists and ethologists have studied for a long time, namely how to teach an animal to accomplish a certain task. Especially for goal-oriented tasks, it is often not obvious to determine which state or action actually led to the goal. Also, if several strategies lead to similar results, it might not be easy to weight them appropriately. Finally, there might sometimes be several different reward functions that all lead to the same result. However, their convergence speed might be different. *Shaping* then is the process that optimizes the parameters of a reward function, and thus the convergence speed of the learning algorithm, without changing the problem setting itself.

On-policy vs. Off-policy Learning A rather technical question is whether the policy that is being learned can already be used during learning itself, or if a different policy has to be used instead. *On-policy* learning methods are those where the learning process is based on the executing of the policy that is being learned, while a *off-policy* methods make no assumption about the policy that is followed during learning. Obviously, any on-policy method is also an off-policy method.

3.1.4 Complexity Results

Given a MDP and a real number K, one can define a decision problem that is associated to computing optimal policies, namely deciding whether there exists a policy π that guarantees an expected accumulated reward of at least K when being executed from initial state s_0. It has been shown that this decision problem can be solved in *polynomial* time with respect to the problem size, given that the transition and the reward function are represented as a table.

Theorem 3.1.1. *The finite-horizon MDP under the finite-horizon criterion, and the infinite-horizon MDP under the discounted infinite-horizon criterion are P-complete.*

Proof. See [PT87]. □

3.1.5 Examples

We present two examples that show how simple decision problems can be encoded withing the Markov decision process framework, which means how appropriate state spaces, transition functions, and reward functions can be defined.

3.1.5.1 Recycling Robot

The goal of the recycling robot [SB98] is to collect soda cans in an office environment. The robot can move in order to detect cans, or return to its docking stating and recharge its batteries. It can

also wait at some place so that people can bring him empty cans. The state of the system can be described by the state of the battery, which can be high s_h or low s_l. The robot has three actions, a_c collect cans, a_w wait, and a_b charge the battery. If the robot decides to look for cans, its battery level remains high with probability $\alpha : \mathcal{T}(s_h, a_c, s_h) = \alpha$ and therefore $\mathcal{T}(s_h, a_c, s_l) = 1-\alpha$. If the battery level is low, the robot can continue searching for cans with probability $\mathcal{T}(s_l, a_c, s_l) = \beta$. With probability $1 - \beta$ however, the battery is depleted and the robot has to be rescued and brought back to its docking station : $\mathcal{T}(s_l, a_c, s_h) = 1 - \beta$. The wait action and the recharging of the battery are deterministic : $\mathcal{T}(s_h, a_w, s_h) = 1$, $\mathcal{T}(s_l, a_w, s_l) = 1$, $\mathcal{T}(s_l, a_b, s_h) = 1$. If the robot is searching for cans, a positive reward $\mathcal{R}_c$ is produced, which corresponds to the expected number of cans the robot will collect. If the robot is waiting, it is given a positive reward of $\mathcal{R}_w < \mathcal{R}_c$. Recharging the battery does not incur any cost, but a penalty of -3 is given to the robot each time it has to be rescued because its batteries are depleted.

3.1.5.2 Grid World

Another class of problems that easily fit the MDP formalism are simple grid worlds, for example the one presented in [RN95]. It consists in a robot that navigates in a two dimensional environment while searching for a goal state, which produces a reward of $+1$, and avoiding a forbidden state with associated reward of -1 (Figure 3.4). Each one of the four possible actions *north*, *east*, *south*, and *west* incurs a cost of -0.04. The system has 11 states, given by the position (x, y) of the robot. The reward function for this example also takes the successor state into account. It can be summarized as follows : $\mathcal{R}(s, a, (4, 3)) = 0.96$, $\mathcal{R}(s, a, (4, 2)) = -1.04$, and $\mathcal{R}(s, a, s') = -0.04$ for each state $s' \neq (4, 3)$ and $s' \neq (4, 2)$. The transitions are probabilistic and succeed in 80% of the cases. In 20% of the cases however, the robot will transition to one of the two neighboring cells. The *north* action, executed in state $(3, 1)$ for example leads the agent to state $(3, 2)$ with probability 0.8, to state $(2, 1)$ with probability 0.1, and finally to state $(4, 1)$ with probability 0.1 as well. An optimal policy and the associated value function for the problem are shown in Figure 3.4.

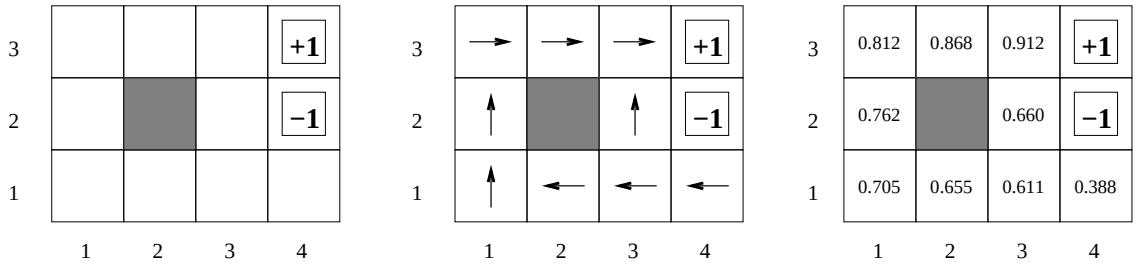

FIG. 3.4 – The initial grid world, an optimal policy, and the associated state value function.

3.2 Partially Observable Markov Decision Processes

The optimal decision rule for MDPs is commonly based on the real state of the system. Unfortunately, this state cannot always be determined with certainty. Sometimes, such as in medical applications for example, it is even impossible to access this state at all [Hau97b]. The only information that is available to the decision maker during execution is a noisy observation or an unreliable signal, which can help him inferring the underlying state. Such problem settings can

be described by an extension of the MDP formalism, the *partially observable Markov decision process* (POMDP).

3.2.1 The POMDP Formalism

The POMDP formalism adds to the MDP the notion of *observations*. Observations are stochastic, and they are generated during each state transition according to an observation function. We now introduce the POMDP formalism for optimal decision making in partially observable environments.

Definition 3.2.1 (POMDP). *A* partially observable Markov decision process *POMDP is defined as a tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$*, where*

- $\mathcal{S}$ *is a finite set of states* $s \in \mathcal{S}$*,*
- $\mathcal{A}$ *is a finite set of actions* $a \in \mathcal{A}$*,*
- $\mathcal{T}(s, a, s')$ *is the probability of transitioning from state* s *to state* s' *after execution of action* a*,*
- $\mathcal{R}(s, a)$ *is the reward produced by the system when action* a *is executed in state* s*,*
- Ω *is a finite set of observations* $o \in \Omega$*,*
- $\mathcal{O}(s, a, o, s')$ *is the probability of observation* o *when action* a *is executed in state* s *and the systems transitions to state* s'*.*

The process restricted to $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *is often called the* underlying MDP *of the POMDP.*

Similar to the MDP model, there exist some variants of this definition. Hauskrecht for example distinguishes between *backward triggered* and *forward triggered* observations, depending on whether an observation at time t is related to the previous state or the new state [Hau97b].

Contrary to the current state, which is a sufficient information for the optimal control of a MDP, the current observation does not necessarily verify the Markov property anymore in the case of a POMDP. This means that the previous solution methods and optimality equations defined over system states cannot directly be applied to the observations of a POMDP. It is however possible to show that a modified description of the process leads to another Markov property, which states that the probability of the system being in some state s solely depends on the *probability distribution over states* at the previous time step. Such a distribution is often called the *belief* about the underlying state, and is denoted $\boldsymbol{b}$. We define $\boldsymbol{b_t}(s)$ as the probability of the system being in state s at time t of the execution :

$$\boldsymbol{b_t}(s) = P(s_t = s | s_0, a_0, o_0, \ldots, a_{t-1}, o_{t-1}) \tag{3.22}$$

The associated belief vector is

$$\boldsymbol{b_t} = \langle \boldsymbol{b_t}(s_1), \boldsymbol{b_t}(s_2), \ldots, \boldsymbol{b_t}(s_{|\mathcal{S}|}) \rangle \tag{3.23}$$

Its update after action a and observation o can easily be derived from Bayes' formula :

$$P(s'|o, \boldsymbol{b}, a) P(o|\boldsymbol{b}, a) = P(o|s', \boldsymbol{b}, a) P(s'|\boldsymbol{b}, a) \tag{3.24}$$

$$P(s'|o, \boldsymbol{b}, a) = \frac{P(o|s', \boldsymbol{b}, a) P(s'|\boldsymbol{b}, a)}{P(o|\boldsymbol{b}, a)} \tag{3.25}$$

and thus explicitly

$$\boldsymbol{b_{t+1}}(s') = \frac{\sum_{s \in \mathcal{S}} \boldsymbol{b_t}(s) \Big[\mathcal{T}(s, a_t, s') \mathcal{O}(s, a_t, o_t, s') \Big]}{P(o|\boldsymbol{b}, a)} \tag{3.26}$$

where $P(o|\boldsymbol{b}, a)$ is a normalizing factor

$$P(o|\boldsymbol{b}, a) = \sum_{s\in\mathcal{S}} \sum_{s'\in\mathcal{S}} \boldsymbol{b_t}(s) \left[\mathcal{T}(s, a_t, s')\mathcal{O}(s, a_t, o_t, s')\right] \tag{3.27}$$

Equation (3.26) shows that the belief distribution is indeed a markovien information. It has finally been shown by Astrom [Ast65] that the belief state is a sufficient statistic for the history of the system, namely the sequence of its states, actions, and observations. It is therefore possible to consider a POMDP as a MDP with a continuous state space, the *belief-state MDP*.

Definition 3.2.2 (Belief-state MDP). *Given a POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}\rangle$, the associated* belief-state MDP *is defined by $\langle \overline{\mathcal{S}}, \overline{\mathcal{A}}, \overline{\mathcal{T}}, \overline{\mathcal{R}}\rangle$, where*

- *$\overline{\mathcal{B}} = \Delta\mathcal{S}$ is the set of belief distributions over states, also called the* belief space*,*
- *$\overline{\mathcal{A}} = \mathcal{A}$ is the set of actions,*
- *$\overline{\mathcal{T}}(\boldsymbol{b}, a, \boldsymbol{b'})$ is the transition function, defined as*

$$\overline{\mathcal{T}}(\boldsymbol{b}, a, \boldsymbol{b'}) = \sum_{s\in\mathcal{S}} \sum_{s'\in\mathcal{S}} \sum_{o\in\Omega} \boldsymbol{b}(s)\mathcal{T}(s, a, s')\mathcal{O}(s, a, o, s')\boldsymbol{b'}(s'),$$

- *$\overline{\mathcal{R}}(\boldsymbol{b}, a)$ is the reward function, defined as*

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

The Bellman equation of this MDP thus becomes

$$V^{t+1}(\boldsymbol{b}) = \max_{a\in\overline{\mathcal{A}}} \left[\overline{\mathcal{R}}(\boldsymbol{b}, a) + \sum_{\boldsymbol{b'}\in\overline{\mathcal{B}}} \overline{\mathcal{T}}(\boldsymbol{b}, a, \boldsymbol{b'})V^t(\boldsymbol{b'})\right] \tag{3.28}$$

The primary obstacle to solving the belief-state MDP is the infinite state space $\overline{\mathcal{B}}$. Each Bellman update would eventually have to require an infinite number of operations. Smallwood and Sondik however could show that the value function at each iteration of the Bellman update can always be represented by finite means [SS73], and that the Bellman operator can thus be applied in finite time. This was a major breakthrough in POMDP theory.

Theorem 3.2.1 (Piecewise Linearity and Convexity of the Value Function). *If V^i denotes a piecewise linear and convex value function, then the Bellman operator conserves this property, and the value function $V^{i+1} = HV^i$ is also piecewise linear and convex.*

In particular, the horizon 1 value function is always piecewise linear and convex, since it is nothing more than the averaged immediate reward for the given state distribution. This guarantees, together with the previous theorem, that the value function for any finite-horizon POMDP, which can be computed through a finite number of applications of the Bellman operator, can always be presented by a finite number of parameters. Smallwood and Sondik's theorem lay the ground for a large number of POMDP solution methods.

Optimal finite-horizon policies for POMDPs can be represented as decision trees, also called *policy trees*, and noted q. They map local histories of observations to actions [KLC98]. An example for such a policy tree is given in Figure 3.5. It describes a situation where the first action should be action a. If the following observation is o_1, then the next action to execute should be a ; if the following observation is o_2, then the next action should be b, and so forth. Evaluating a policy tree q given an initial state s_0 can be done by simulating all possible sequences of actions, and by considering all corresponding sequences of observations and underlying states.

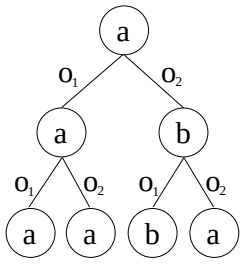

FIG. 3.5 – A policy tree of depth 3 for a problem with 2 observations o_1, o_2, and 2 actions a, b.

3.2.2 Solving POMDPs Using Dynamic Programming

The value function of a single policy q can always be represented as a $|\mathcal{S}|$-dimensional vector, the so called $\boldsymbol{\alpha}$-vector $\boldsymbol{\alpha} = \langle V(s_1, q), \ldots, V(s_{|\mathcal{S}|}, q)\rangle$. It represents the value of the policy at the corners of the belief space. The value of any intermediate belief state can be obtained by simple linear interpolation :

$$V(\boldsymbol{b}, q) = \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V(s, q) \tag{3.29}$$

Given a set of policies Q, the optimal value for any belief state $\boldsymbol{b}$ can be determined through maximizing over all available policies :

$$V(\boldsymbol{b}) = \max_{q \in Q} V(\boldsymbol{b}, q) = \max_{q \in Q} \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V(s, q) \tag{3.30}$$

According to Smallwood and Sondik's theorem, the set of policies - and therefore the set of $\boldsymbol{\alpha}$-vectors - that is needed to represent the optimal value function is always finite. An example for such a value function is represented in Figure 3.6. For each belief state $\boldsymbol{b}$, there is one optimal

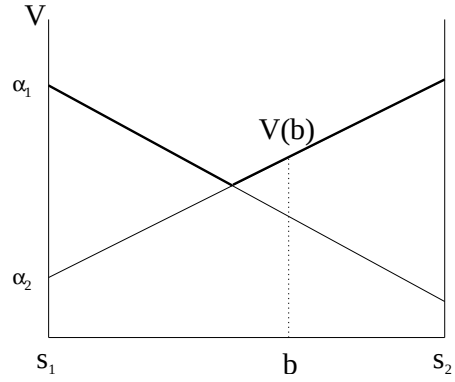

FIG. 3.6 – The value function of a POMDP with two states s_1 and s_2, represented by two vectors $\boldsymbol{\alpha_1}$ and $\boldsymbol{\alpha_2}$. The action associated with vector $\boldsymbol{\alpha_2}$ corresponds to the optimal action in belief state $\boldsymbol{b}$.

policy, and one maximal $\boldsymbol{\alpha}$-vector that dominates all other vectors. A fundamental question for the optimal solution of a POMDP then is : is it possible to identify those $\boldsymbol{\alpha}$-vectors that are suboptimal over the entire belief space, and that can thus be pruned away ?

Dominated Policies and Pruning A policy that is optimal in a given belief state is said to be the *dominating* policy in that state. Similarly, a policy that is suboptimal in a belief state is said to be *pointwise dominated*. One interesting question concerns identifying those policies that are dominated over the entire belief space. Such a policy is called a *dominated policy*. The $\boldsymbol{\alpha}$-vector associated with this policy is called a *dominated vector*. A vector that is not entirely dominated is said to be *useful*. An example of a value function with two useful vectors and one dominated vector is shown in Figure 3.7. It is important to emphasize on the fact that each

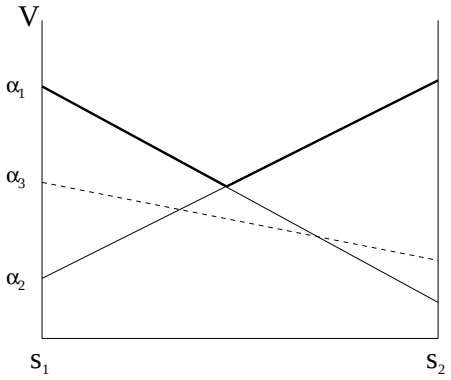

FIG. 3.7 – The value function of a POMDP with two states s_1 and s_2, represented by three vectors $\boldsymbol{\alpha_1}$, $\boldsymbol{\alpha_2}$, and $\boldsymbol{\alpha_3}$. The vector $\boldsymbol{\alpha_3}$ is entirely dominated and can be pruned without affecting the representation of the optimal value function.

dominated $\boldsymbol{\alpha}$-vector can be *pruned* from the set of policies without affecting the optimal value function. It has furthermore been shown that a dominated policy can never be part of an optimal policy, which means that a dominated policy can never appear as a sub-tree of an optimal policy. Identifying and pruning dominated policy vectors constitutes the major computational effort in dynamic programming for POMDPs. Formally, an $\boldsymbol{\alpha}$-vector is dominated, if the associated policy q verifies

$$(\forall \boldsymbol{b})(\exists \tilde{q} \neq q) \qquad \text{such that} \qquad V(\boldsymbol{b}, q) \leq V(\boldsymbol{b}, \tilde{q}) \tag{3.31}$$

This leads directly to the linear program of Figure 3.8, first introduced by Monahan [Mon82], to test if a policy q is dominated or not. If the result of that program is negative or equal to zero

maximize	ϵ	
subject to	$V(\boldsymbol{b}, \tilde{q}) - V(\boldsymbol{b}, q) \geq \epsilon,$	$(\forall \tilde{q} \neq q)$
with	$\sum_{s \in \mathcal{S}} \boldsymbol{b}(s) = 1, \quad \boldsymbol{b}(s) \geq 0,$	$(\forall s \in \mathcal{S})$

FIG. 3.8 – Linear program for identifying dominated policies.

($\epsilon \leq 0$), then the policy q, and the associated $\boldsymbol{\alpha}$-vector, can be pruned.

We have seen that there exists a correspondence between the set of vectors $\boldsymbol{\alpha} \in \Gamma$ that constitute the value function at horizon t, and the set of policy trees $q \in Q$ at depth t. Computing the optimal value function by dynamic programming is thus equivalent to constructing the associated

set of policies. As it is the case for MDPs, this construction proceeds incrementally horizon by horizon.

Suppose the existence of an optimal value function for horizon t, represented by a set of $\boldsymbol{\alpha}$-vectors Γ^t, or equivalently to a set of policy trees Q^t. At the beginning of the algorithm, this set will be constituted of the horizon-1 policies, which means the set of actions $\mathcal{A}$. Building the value function for horizon $(t+1)$ means first enumerating all possible policies for that horizon. For constructing a new policy tree q^{t+1}, one first choses an action $a \in \mathcal{A}$ to be affected to the root, and then $|\Omega|$ policies $q_i^t \in Q^t$ to be associated to each observation. There are a total of $|\mathcal{A}||Q^t|^{|\Omega|}$ policies to be generated, and we call *exhaustive generation of policies* the process that constructs the entire set of these policies. It is shown in Algorithm 18. The second step consists in pruning all those policies of the newly constructed set Q^{t+1} that are completely dominated. This step usually involves linear programming. A general dynamic programming approach for POMDPs is

Algorithm 18 Exhaustive Generation of Policies - `ExhaustiveGeneration()`

Require: A set of policies Q^t for horizon t
Ensure: A set of policies Q^{t+1} for horizon $(t+1)$
1: **for all** action $a \in \mathcal{A}$ **do**
2: **for all** selection of $|\Omega|$ policies q_i^t from Q^t with $1 \leq i \leq |\Omega|$ **do**
3: Construct a new policy q^{t+1} such that :
4: /* Assign an action with the new root node */
5: $\alpha(q^{t+1}) \leftarrow a$
6: /* Assign subtrees for all observations */
7: $q^{t+1}(o_i) \leftarrow q_i^t, \qquad (\forall o_i \in \Omega)$
8: /* Add the new policy tree to the result set */
9: $Q^{t+1} \leftarrow Q^{t+1} \bigcup \{q^{t+1}\}$
10: **end for**
11: **end for**

summarized in Algorithm 19. It is important to note that the construction of the policy trees, and thus the construction of the value function, is a *bottom-up* approach, which means that the first policies that are generated are those that will be executed last. At each iteration, a new set of policies is constructed, where each policy contains the policies of the previous iteration step as subtrees. The algorithm is in fact the POMDP extension of the backward induction algorithm for MDPs (Algorithm 15). The main bottleneck of the basic dynamic programming algorithm

Algorithm 19 Dynamic Programming for POMDPs - `DPforPOMDPs()`

Require: A POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$ and a horizon T
Ensure: A set of optimal policies for horizon T
1: Initialize $Q^1 \leftarrow \mathcal{A}$
2: **for** $t = 2$ **to** $t = T$ **do**
3: /* Exhaustive generation of policies */
4: $Q^t \leftarrow$ `ExhaustiveGeneration`(Q^t)
5: /* Pruning */
6: Prune every policy $q \in Q^t$ if $(\forall \boldsymbol{b})(\exists \tilde{q} \neq q)$ such that $V(\boldsymbol{b}, q) \leq V(\boldsymbol{b}, \tilde{q})$
7: **end for**

(Algorithm 19) consists in the necessity of first generating the exhaustive set of policies before

beginning the process of pruning. There exist smarter techniques that interleave the generation of policies with the pruning step.

The Witness Algorithm Kaelbling et al. have introduced an improvement of the naive dynamic programming approach presented above, namely the *witness algorithm* [KLC98]. Instead of directly generating the exhaustive set of policies Q^t, the witness algorithm constructs, for each action a, a set Q_a^t of policies that have action a at their root. The set Q^t can then be obtained by unifying all sets Q_a^t. The main idea is the following : instead of first generating all possible policies for Q_a^t, a direct search for a minimal representation of Q_a^t is conducted. This is done by testing, after each policy generation, if the set Q_a^t is already an optimal representation of the value function, or if there still exists a belief state $\boldsymbol{b}$ at which the value function can be improved by adding a new policy. Each such belief state is called a *witness*, since it witnesses the fact that Q_a^t is not yet an optimal representation of the value function. Identifying a witness point can

maximize	ϵ	
subject to	$V(\boldsymbol{b}, q_{new}) - V(\boldsymbol{b}, \tilde{q}) \geq \epsilon,$	$(\forall \tilde{q} \neq q)$
with	$\sum_{s \in \mathcal{S}} \boldsymbol{b}(s) = 1, \quad \boldsymbol{b}(s) \geq 0,$	$(\forall s \in \mathcal{S})$

FIG. 3.9 – Linear program for identifying witness points.

be done by solving the linear program shown in Figure 3.9, where q_{new} denotes any candidate policy that is not yet in Q_a^t. The witness loop has thus to be repeated for all possible q_{new}.

Incremental Pruning Even more sophisticated algorithms try to split up the process of generating new policies and pruning dominated vectors. In general, the earlier the pruning step can be executed, the faster the algorithm can proceed. The *incremental pruning* technique splits up the generation of the set Q_a^t into several subsets $Q_{a,o}^t$, one for each observation o. The set Q_a^t can then be obtained by applying the *cross sum* operator :

$$Q_a^t = \bigoplus_{o \in \mathcal{O}} Q_{a,o}^t \tag{3.32}$$

where $A \oplus B = \{\alpha + \beta | \alpha \in A, \beta \in B\}$. Cassandra et al. could show in [CLZ97] that pruning the set Q_a^t obtained from the cross sum of the sets $Q_{a,o}^t$ can be done incrementally in the following way :

$$\texttt{prune}\left(Q_a^t\right) = \texttt{prune}\left(Q_{a,o_1}^t \oplus Q_{a,o_2}^t \oplus \cdots \oplus Q_{a,o_k}^t\right) \tag{3.33}$$

$$= \texttt{prune}(\ldots \texttt{prune}(\texttt{prune}(Q_{a,o_1}^t \oplus Q_{a,o_2}^t) \oplus Q_{a,o_3}^t) \cdots \oplus Q_{a,o_k}^t) \tag{3.34}$$

The family of incremental pruning algorithms is considered as being one the most efficient ways of solving POMDPs via dynamic programming, and has since continuously been improved (see for example [FZ04]).

3.2.3 Solving POMDPs Using Heuristic Search

Constructing an optimal policy tree can also be interpreted as the search for a conditional path in the space of possible policies. The associated search tree for POMDPs is an example of an

AND/OR tree [Pea90], in which case the construction of the optimal policy is *top-down*. For each node, which contains an action, all of the $|\Omega|$ possible child nodes have to be considered (AND part), but for each one of these children, only one next action has to be associated (OR part). An example of such an AND/OR search tree for POMDPs is shown in Figure 3.10. Finding the

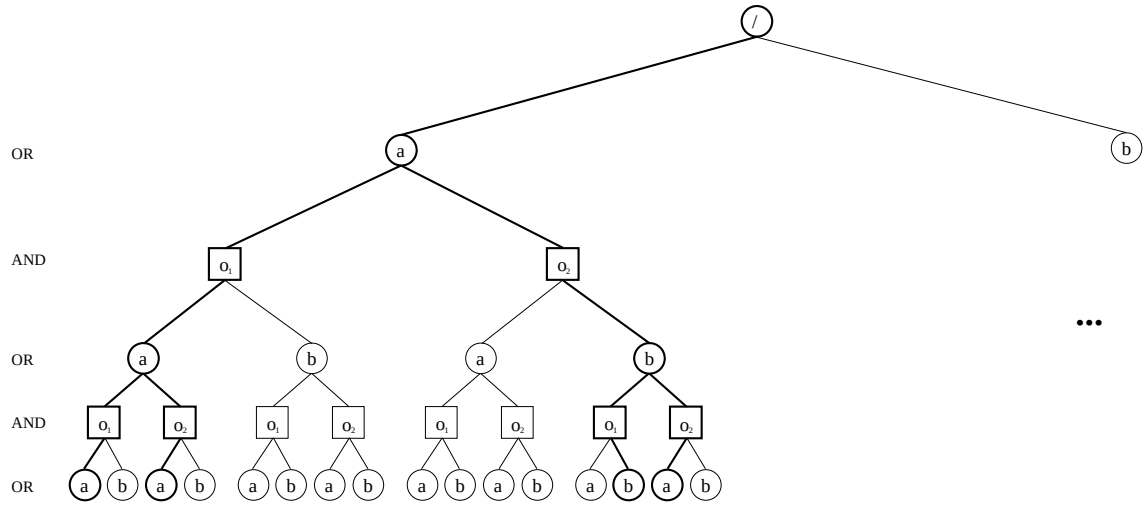

FIG. 3.10 – An extract of an AND/OR search tree for solving POMDPs. The nodes a and b are OR nodes and correspond to actions, whereas the nodes o_1 and o_2 are AND nodes and correspond to observations. A solution path is indicated in bold, and it represents the same policy as the one shown in Figure 3.5

optimal conditional path in an AND/OR tree can be done through heuristic search techniques such as the A* algorithm [Nil80, Pea90]. A* is a best-first search algorithm, combined with a heuristic evaluation function that helps selecting the current best policy based on an easily computable value estimate. It is guaranteed to find the optimal solution with minimal effort. Evaluating a leaf λ at depth t of an A* search tree means computing the function

$$F^T(\boldsymbol{b_0}, \lambda) = G^t(\boldsymbol{b_0}, \lambda) + H^{T-t}(\boldsymbol{b_0}, \lambda) \tag{3.35}$$

where $\boldsymbol{b_0}$ is the initial belief state, G is the exact value of the path from the root until leaf node λ, H is a heuristic estimate for the optimal value of the remaining path that starts from λ, and T is the problem horizon [Pea90]. The heuristic function H is said to be *admissible*, if it overestimates the value of the optimal path. This property guarantees the convergence of the algorithm : if a solution is found that outperforms all current heuristic estimates, then it also outperforms all possible underlying paths covered by that heuristic, since they would lead to even worse evaluations.

Each one of the leaves λ of the search tree can be associated to one sequence of actions and observation $\lambda \equiv (a_1, o_1, \ldots, a_{t-1}, o_{t-1}, a_t)$, and we denote $\boldsymbol{b}_{\boldsymbol{b_0},\lambda}$ the belief state associated to leaf λ, given initial belief $\boldsymbol{b_0}$. Evaluating the path from the root to leaf node λ is equivalent to determining the value of the associated Markov chain :

$$V(s, (a_1, o_1, a_2, o_2, \ldots, a_t)) = \mathcal{R}(s, a_1) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a_1, s') \mathcal{O}(s, a_1, o_1, s') V(s', (a_2, o_2, \ldots, a_t)) \tag{3.36}$$

The function G represents the weighted sum of all these evaluations :

$$G^t(\boldsymbol{b_0}, \lambda) = \sum_{s \in \mathcal{S}} \boldsymbol{b_0}(s) V(s, \lambda) \tag{3.37}$$

Whereas G can be determined explicitly, establishing an optimistic estimate of the remaining path in fact means determining an easily computable upper bound for a POMDP. Such a bound can for example be established by considering the underlying MDP process (see Definition 3.2.1). It can be shown that

$$V^{T-t}_{POMDP}(\boldsymbol{b}) \leq \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V^{T-t}_{MDP}(s) \tag{3.38}$$

where V^{T-t}_{POMDP} and V^{T-t}_{MDP} denote the value functions for the POMDP and the underlying MDP, respectively [LCK95, Hau00]. The heuristic function H over the remaining horizon $(T-t)$ and defined as

$$H^{T-t}(\boldsymbol{b_0}, \lambda) = H^{T-t}(\boldsymbol{b}_{\boldsymbol{b_0},\lambda}) = \sum_{s \in \mathcal{S}} \boldsymbol{b}_{\boldsymbol{b_0},\lambda}(s) V^{T-t}_{MDP}(s) \tag{3.39}$$

therefore is admissible by definition. There exist other upper bounds for the value function of a POMDP, and an extended overview can be found in [Hau00]. All of them can be used as a possible heuristic evaluation function to guide the search process.

The application of heuristic search to solving POMDPs has notably been studied by [Was96, GB98], and Algorithm 20 represents a variant of A* for solving finite-horizon problems, with a heuristic function that is based on the underlying MDP and that evaluates as follows :

$$\overline{V}^T(\boldsymbol{b}, \lambda) = \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V(s, \lambda) + \sum_{s \in \mathcal{S}} \boldsymbol{b'}_{\boldsymbol{b},\lambda}(s) V^{T-t}_{MDP}(s) \tag{3.40}$$

Algorithm 20 Heuristic Search for POMDPs - `POMDPAStar()`

Require: A POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$ and a horizon T
Ensure: An optimal policy for the initial belief state $\boldsymbol{b_0}$

Initialize the OPEN list D with an empty node $\emptyset$
repeat
  Chose $\lambda \in D$ such that $(\forall \lambda' \neq \lambda) : \overline{V}(\boldsymbol{b_0}, \lambda') \leq \overline{V}(\boldsymbol{b_0}, \lambda)$
  Build a new policy tree q such that :
  The root of q matches with $\lambda : \alpha(q) \leftarrow \lambda$
  for all $o \in \Omega$ **build** a child :
    $q(o) \leftarrow o$
    for all $a \in \mathcal{A}$ **build** a grandchild :
      $q(o)(a) \leftarrow a$
    end for
  end for
  Replace λ with q
until the current best policy λ has depth T

3.2.4 Infinite-horizon POMDPs

Extending the previous dynamic programming and heuristic search techniques to infinite-horizon POMDPs involves addressing two issues : the policy representation, and the value function update. Policies for infinite-horizon problems can obviously not be represented as trees of only finite depth. At the same time, it is difficult to imagine trees of infinite depth. This is why alternative decision structures, such as finite state automata, are frequently being used as policy representations. An example for such a finite state controller is shown in Figure 3.11. It should

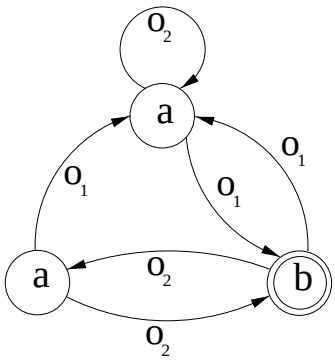

FIG. 3.11 – A deterministic finite state controller with 3 nodes for a problem with 2 actions (a and b) and 2 observations (o_1 and o_2).

be noted that a finite state controller may not be sufficient to represent an optimal policy since it is bounded in size. Determining its optimal size, or determining an error bound for a given size, are problems that do not seem to be adequately solvable yet.

The value function update can be extended much more easily to infinite-horizon problems. The extension is similar to the POMDP value iteration algorithm in that it basically requires an infinite application of the dynamic programming operator, and the introduction of a discount factor. The Bellman equation for the infinite-horizon POMDP can be written as follows

$$V^*(\boldsymbol{b}) = \max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) \left[R(s,a) + \gamma \sum_{o \in \Omega} O(s,a,o) V^*(\boldsymbol{b}'_{a,o}) \right] \tag{3.41}$$

where $\boldsymbol{b}'_{a,o}$ is the posterior belief over states after action a and observation o. It is the POMDP equivalent to the Bellman equation (3.16) for fully observable MDPs, and it can be approximated by an infinite loop of the above dynamic programming algorithm.

3.2.5 Approximations

Because of the worst case complexity of the POMDP model, there is a boundary in problem size beyond which optimal planning algorithms currently fail to succeed : they either run out of time or out of memory. This is why approximating the optimal POMDP solution has gained in interest. Approximating the solution of a POMDP can either be done in policy space or in value function space, and the latter one is usually preferred because it enables to express error bounds on the optimal value function.

3.2.5.1 Approximations Based on Value Function Bounds

Hauskrecht states in his survey paper [Hau00] that "*perhaps the simplest way to approximate the value function for a POMDP is to assume that states of the process are fully observable*". This means that the value function is only determined at the corners of the belief simplex, and that the value of each intermediate belief state can be approximated as :

$$V(\boldsymbol{b}) \leq \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V_{MDP}(s) \tag{3.42}$$

This approximation, often called the *MDP approximation*, has been suggested by Lovejoy [Lov91], and V_{MDP} denotes the optimal value function of the underlying MDP. Obviously, this approach does not take into account partial observability at all, and the resulting value function thus is entirely linear between two states (see the left part of Figure 3.12). A slightly more accurate

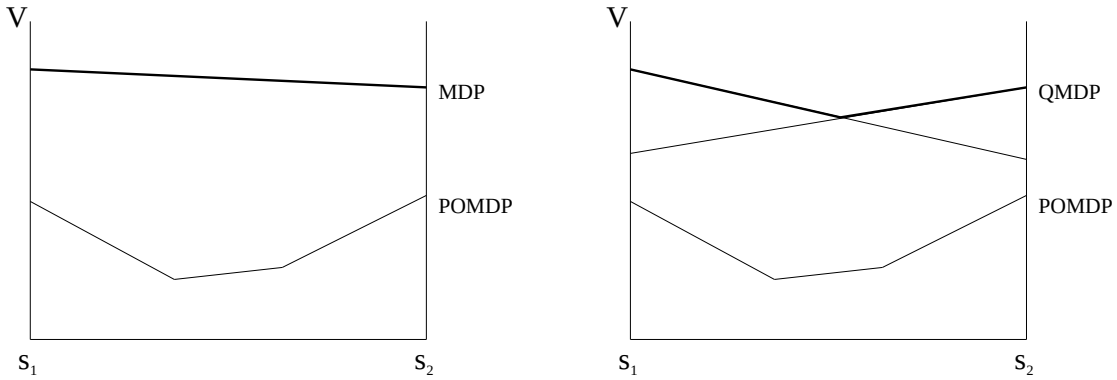

FIG. 3.12 – Two value function approximations for POMDPs, based on the underlying fully observable MDP. Left : the MDP approximation - Right : the QMDP approximation.

variant of the MDP approximation uses Q-functions in order to approximate the POMDPs value function. It is therefore called the *QMDP approximation* [LCK95]. It restricts the choice of actions, that can be selected in any given belief state, to a single one (whereas the previous approach assumed that for each state a different action choice was possible) :

$$V(\boldsymbol{b}) \leq \max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) Q_{MDP}(s, a) \tag{3.43}$$

An example of a resulting value function with the QMDP approach is shown in the right part of Figure 3.12.

Both the MDP and the QMDP approximations are upper bounds to the optimal POMDP value function. It turns out that a lower bound, albeit not a very accurate one, can be obtained by assuming an *unobservable MDP*. Hauskrecht shows that an update rule for the unobservable MDP can equally be established [Hau00], but since it remains an NP-hard problem, and since we do not need any lower bounds for POMDPs, we will not present it in more detail.

3.2.5.2 Point-based Dynamic Programming

The dynamic programming approach presented in Section 3.2.2 relies heavily on solving linear programs to identify dominated policy vectors. These are essentially continuous optimization

problems. The value function however always admits a finite representation, at least for finite-horizon problems. The question thus arises if there is not a finite approach that allows to identify these vectors one by one.

As we said earlier, Lovejoy [Lov91] was maybe the first one to introduce an approximation method for evaluating the POMDP value function based on a discretization of the belief space using grids. The value function is computed at the grid points only. Its value between these grid points is then interpolated, and the complexity of this approach is thus independent of the inherent structure of the value function. It solely depends on the resolution of the grid. The resolution obviously influences the accuracy of the approximation. The drawback of this approach consists in the uniform distribution of the grid points. It can be assumed that useful real-world belief states are not uniformly distributed, and a grid based approach makes it difficult to adapt to those regions of the belief space that are more likely to be visited than others. This insight has let Pineau et al. to work on a variant of the grid-based approach, which explicitly determines a set of *relevant belief states* by forward sampling real world trajectories [PGT03]. Their *point-based value iteration* algorithm starts with an initial belief, or an initial distribution over states, and first computes a set of reachable posterior beliefs, given all possible actions, all possible observations, the transition function, and the observation function of the system. The value function update is then performed at those reachable belief states only. An example of such a simulation of belief states is given in Algorithm 21. The procedure is an iterative improving of a set of belief states, initialized with the initial belief of the start state distribution. It computes, for each belief state and each action, a set of possible successor states B_{a_i}, but keeps only the successor belief that is the farthest away from the beliefs already in B. For finite-horizon problems, executing the procedure t times can be interpreted as sampling all belief states reachable with a horizon t policy.

Algorithm 21 Sampling of relevant belief states - `BeliefSampling()`

Require: A set of belief states B
Ensure: An improved set of belief states B'
1: $B' \leftarrow B$
2: **for all** belief state $\boldsymbol{b} \in B$ **do**
3: **for each** action $a_i \in \mathcal{A}$ **do**
4: Sample a state s according to $\boldsymbol{b}$
5: Sample an observation o according to s, a_i, and $\mathcal{O}$
6: $B_{a_i} \leftarrow \boldsymbol{b}_{a_i} = \boldsymbol{b'}$ through Equation (3.26)
7: **end for**
8: $B' \leftarrow B' \bigcup \left\{ \arg \max_{\boldsymbol{b}_{a_i} \in B_{a_i}} \left(\max_{\boldsymbol{b} \in B} ||\boldsymbol{b}_{a_i} - \boldsymbol{b}||_1 \right) \right\}$
9: **end for**

The approach developed by Pineau et al. considers the optimal policy at each belief point instead of its value. This means that the properties of piecewise linearity and convexity of the value function are preserved. An example for two value functions, one with a useful set of belief samples, and one with a suboptimal set of belief points, is presented in Figure 3.13. Algorithm 22 summarizes the core of the point-based dynamic programming approach for solving finite-horizon POMDPs. It has been shown to be quite useful in practice, and has led to several extensions [PGT03, SS05, SV05]. Since it does not require dynamic programming, and since it permits to approximate the value function consistently, it is able to solve much larger problems.

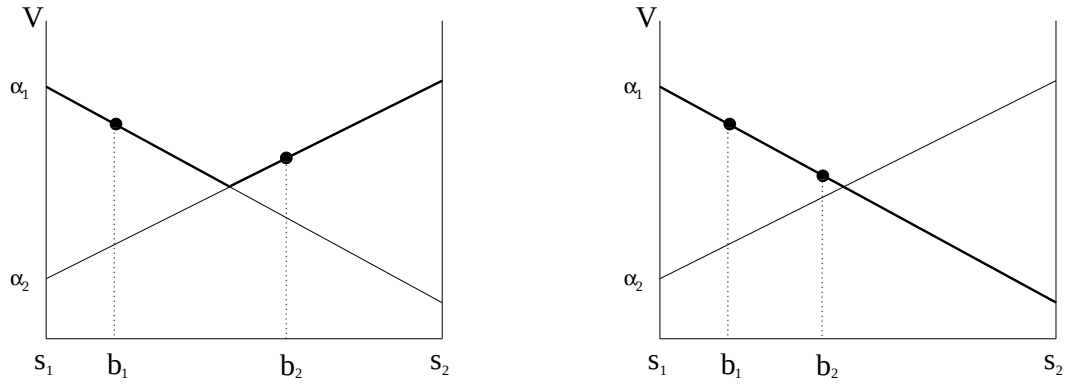

FIG. 3.13 – Evaluation of a POMDP value function by two different belief sets. Left : the belief points b_1 and b_2 are well placed to identify the two useful $\boldsymbol{\alpha}$-vectors and associated policies. - Right : the two belief points identify the same policy vector, the second policy remains unidentified. For those belief points that fall into this region, the point-based approximation is not optimal, since it retains vector $\boldsymbol{\alpha_2}$, although vector $\boldsymbol{\alpha_1}$ is the optimal one.

3.2.6 Complexity Results

POMDPs are significantly harder to solve than MDPs : the complexity goes up from polynomial to exponential. Given a POMDP, a real number K, and an initial state s_0, one defines the associated decision problem to solving a POMDP as determining whether there exists a policy that guarantees an expected accumulated reward of at least K when being executed from state s_0.

Theorem 3.2.2. *The finite-horizon POMDP under the finite-horizon criterion is PSPACE complete. The infinite-horizon POMDP under the discounted infinite-horizon is undecidable.*

Proof. See [MHC03]. □

In general, solving a finite-horizon POMDP optimally requires exponential time with respect to the problem size. When moving to infinite-horizons, the problem changes its complexity class and becomes in general intractable. This is why solving infinite-horizon POMDPs is usually restricted to a certain class of policies, such as finite automata.

3.2.7 Examples

In the following section, we will present two common example problems of sequential decision making under partial observability that can be modeled using the POMDP formalism.

3.2.7.1 The Tiger Problem

The tiger problem, introduced by [Kae93], simulates an agent that faces two closed doors. Behind one of the doors is a tiger, whereas behind the other door is a large reward. If the agent opens the door with the tiger, it perceives a large negative penalty. Its goal thus is to avoid the tiger and to open the door with the large positive reward. To gain information about the position of the tiger, the agent can listen at the doors. Listening incurs a cost, and it is also not precise. With some probability, the agent can get the impression that the tiger is behind the wrong door.

Algorithm 22 Point-based dynamic programming for POMDPs - `PointBasedDP()`

Require: A POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$, a horizon T, and an initial belief b_0
Ensure: A set of - possibly suboptimal - policies for horizon T

```
Initialize Q^1 ← A
Initialize B ← {b_0}
Apply T times B ← BeliefSampling(B)
for t = 2 to t = T do
    /* Exhaustive policy generation */
    Q̃^t ← ExhaustiveGeneration(Q^{t-1})
    /* Point-based policy updates */
    for all belief state b ∈ B do
        Q^t ← Q^t ⋃ { arg max_{q∈Q̃^t} V(b, q) }
    end for
end for
```

Nevertheless, listening is the only source of information to determine the position of the tiger.

The tiger problem can be formulated as a POMDP. The state can be represented as a binary variable, indicating whether the tiger is behind the left door s_l or the right door s_r. There are three actions, a_l, a_r, and a_s, which represent opening the left door, opening the right door, and listening. The reward is $+10$ if the correct door is opened, whereas a reward of -100 is given if the door with the tiger is opened. The cost for listening is -1. There are two possible observations, namely hearing the tiger behind the left door o_l, and hearing it behind the right door o_r. The initial state distribution places the tiger behind each door with equal probability, and the system is reset to this state each time the agent opens a door. Listening does not change the state of the system, which means that $\mathcal{T}(s, a_s, s') = 1$ if $s = s'$, and $\mathcal{T}(s, a_s, s') = 0$ otherwise. The system transitions to s_l with probability 0.5, and to s_r with probability 0.5 each time a door is opened : $\mathcal{T}(s, a, s_l) = 0.5$ et $\mathcal{T}(s, a, s_r) = 0.5$ for all states $s \in \{s_g, s_d\}$ and all actions $a \in \{a_l, a_r\}$. The observation function is probabilistic, and the observations are accurate only in 85% of all cases : $\mathcal{O}(s_l, a_s, o_l, s_l) = 0.85$, $\mathcal{O}(s_l, a_s, o_r, s_l) = 0.15$, $\mathcal{O}(s_r, a_s, o_r, s_r) = 0.85$, and $\mathcal{O}(s_r, a_s, o_l, s_r) = 0.15$.

3.2.7.2 Managing Patients with Ischemic Heart Disease

POMDP theory has also been applied to medical problems, such as managing patients that suffer from ischemic heart disease [Hau97a, Hau97b]. We do not want to go into the medical details of the problem, but rather emphasize on how such a problem can be described using Markov decision processes. The markovian state description of the patient includes several factors, for example the state of its coronary artery disease, its ischemia level, the history of its myocardial infarction, the history if its coronary artery bypass surgery, and others. These factors however are not always fully accessible. The observation space includes the history of the patient's myocardial infarction, but also data such as its resting EKG, results from stress tests or angiogram investigations, or simple chest pain. The actions available to the decision maker, in this case the doctor, include a medical treatment, a stress test, an angiogram investigation, but also the option of doing nothing at all. The description is taken from [Hau97a], and it is not exhaustive. It is just meant to show how difficult the establishing of an appropriate stochastic model can be, especially if the

underlying transition and observation probabilities are particularly hard to estimate.

3.3 Other Markov Models for Sequential Decision Making

The Markov decision process framework is quite general, and we have seen that even problems that are not inherently markovian can be transformed into problems that indeed verify the Markov property. It is however not the only model for sequential decision making, and the algorithms we presented above are not the only techniques to solve these problems. In this section, we want to briefly mention some related models that we do not discuss in the thesis, but that might be of great interest.

3.3.1 Semi-Markov Decision Processes

All of the above Markov frameworks assume atomic actions that that the same amount of time to execute. This is obviously a simplification, and one can easily imagine problems where actions have different durations. One way of dealing with this issue is to introduce new states, in which the system will remain for the duration of the action. Another way is to model duration explicitly within the model. The *semi-Markov decision process*, or SMDP, allows for actions to take multiple time steps [SPS99]. This enables for example integrating macro-actions, sometimes called *options*, into the planning process, and to establish hierarchies of abstraction [Die00, MB01]. Planning can thus be decomposed, for example into higher level strategic planning, and lower level action planning.

3.3.2 Predictive State Representation

The *predictive state representation* framework PSR is an equivalent approach to solving partially observable Markov decision problems. It was introduced by Littman, Sutton, and Singh [LSS01, SJR04] and represents states by "*multi-step, action-conditional predictions of future observations*". PSRs are based on the notion of *tests*, where a test is a sequence of actions and observations $(a_1, o_1, a_2, o_2, \ldots, a_k, o_k)$. The *prediction* of a test is the probability of observing the sequence of observations $(o_1, o_2, \ldots, o_k)$ while executing the sequence of actions $(a_1, a_2, \ldots, a_k)$. The authors could show that PSRs may have some advantages over POMDPs both in compactness and expressiveness, and new learning and planning algorithms for PSRs are currently being developed [SLJ+03].

3.3.3 Bayesian Networks

Bayesian networks with structured conditional probability matrices sometimes allow a more natural way of specifying the effects of actions in dynamic systems. They also help factorizing the state space in an intuitive way, and enable expressing independencies between random variables. This is why they are being studied by the planning community [BP96, Gue03]. We describe a simple example of how a POMDP can be represented as a Bayesian network (taken from [BP96]). An office robot is designed to bring coffee to the office workers. It can do so by crossing the street, and it is rewarded if the user indeed wants coffee *WC* and the robot has coffee *HC*. It is however penalized if it does not have coffee when the user wants some. The robot will get wet *W* if it is raining *R* when it goes for coffee, unless it has an umbrella *U*. The Bayesian network for the action `GetCoffee` is shown in Figure 3.14.

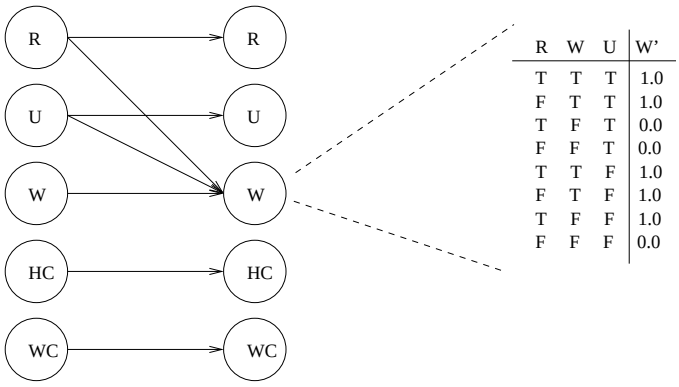

FIG. 3.14 – The Bayesian action network for action `GetCoffee`. Given is also the probability table for whether the robot is wet or not after getting coffee. The example is taken from [BP96].

3.4 Conclusion

The MDP and POMDP formalisms, notions such as value functions, policies, or belief states, and the basic dynamic programming techniques such as the value iteration algorithm, constitute the theoretical foundations of our work. MDPs and POMDPs have been shown to be very general and expressive frameworks, and controllers based on MDPs could help solving complicated tasks such as balancing a pole [MC68] or learning how to swim [Cou02]. We do not seek to hide the fact that the MDP model remains a discrete and sequential model, and that it assumes the Markov property to be verified. Nevertheless, most - if not all - single-agent decision problems can usually be described in one way or another using MDP or POMDP theory.

Chapitre 4

Decentralized Markov Decision Processes

4.1 The DEC-POMDP Model

The most natural extension of the single-agent Markov decision processes to multiple and decentralized decision makers is the *decentralized POMDP* model, introduced by Bernstein et al. in [BZI00] and later in [BGIZ02]. It will constitute the formalism on which our work is based on, and we will present it now in detail.

4.1.1 The Formalism

The DEC-POMDP formalism assumes the presence of multiple decision makers that influence the Markov process simultaneously. Each decision maker, or *agent*, has it own local actions and observations, which cannot be shared with other agents. The transition, observation, and reward functions however depend on the joint action executed by the whole team of agents.

Definition 4.1.1 (DEC-POMDP). *A DEC-POMDP is defined as a tuple* $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$*, where*

- $\mathcal{S}$ *is a finite set of states* $s \in \mathcal{S}$*,*
- $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *is a finite set of joint actions* $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$*, and* $\mathcal{A}_i$ *is the set of actions* a_i *for agent* i*,*
- $\mathcal{T}(s, \boldsymbol{a}, s')$ *is the probability of transitioning from state* s *to state* s' *after execution of joint action* $\boldsymbol{a}$*,*
- $\mathcal{R}(s, \boldsymbol{a})$ *is the reward produced by the system when joint action* $\boldsymbol{a}$ *is executed in state* s*,*
- $\boldsymbol{\Omega} = \Omega_1 \times \cdots \times \Omega_n$ *is a finite set of joint observations* $\boldsymbol{o} \in \boldsymbol{\Omega}$*, and* Ω_i *is the set of observations* o_i *for agent* i*,*
- $\mathcal{O}(s, \boldsymbol{a}, \boldsymbol{o}, s')$ *is the probability of joint observation* $\boldsymbol{o}$ *when joint action* $\boldsymbol{a}$ *is executed in state* s *and the system transitions to state* s'*.*

Similar to an earlier definition in the context of single-agent POMDPs, we define the underlying POMDP *of a DEC-POMDP as the process that treats joint actions as atomic actions, and thus does not take into account the presence of multiple decision makers. The underlying POMDP can be seen as a central entity that collects the observations of all agents, determines the joint action to executed, and then distributes the individual actions back to the agents.*

4.1.2 Policies

Solving a DEC-POMDP means finding a set of local policies, also called a *joint policy*, such that their synchronous but decentralized execution maximizes the associated performance criterion, e.g. maximizes the expected accumulated reward of the system. It should be emphasized that this reward is common to the whole team. There are no individual rewards, which makes the system entirely cooperative. Each agent i executes a local action a_i based on its local policy. The joint action $\boldsymbol{a} = \langle a_1, \ldots, a_n \rangle$ changes the configuration of the system from its current state s to a new state s', and a joint observation $\boldsymbol{o} = \langle o_1, \ldots, o_n \rangle$ is produced. Each agent however has only access to its local component o_i of this observation in order to chose his next action.

We have introduced a special class of history dependent policies for solving partially observable Markov decision processes, namely the class of *decision graphs*. In the case of finite-horizon problems, a decision graph is a *policy tree* (an example is given in Figure 3.5). In the case of infinite-horizon problems, we will be using *finite state controllers* (an example is given in Figure 3.11). We note q_i^t a depth-t policy tree for agent i, and q_i^N a finite state controller with N nodes for agent i. We define a joint policy as follows :

Definition 4.1.2 (Joint Policy). *A* joint policy $\boldsymbol{q}$ *denotes a set of* n *local policies* $\boldsymbol{q} = \langle q_1, \ldots, q_n \rangle$*, such that each local policy* q_i *is associated to, and executed by, the corresponding agent* i*.*

Solving the underlying centralized POMDP that corresponds to the DEC-POMDP would lead to a single policy where the vectors of joint actions and joint observations are represented by atomic actions and atomic observations respectively. The underlying POMDP controller can thus be represented as an omniscient entity that is able to collect all local observations, and to distribute its decision back to all agents.

The major additional difficulty of the DEC-POMDP as compared to the POMDP lies in the partial distributed observability. In fact, agents do not only partially observe the underlying state of the system, but they also have an incomplete view of the local information state of their teammates. Each agent is certainly able to derive an estimation of the system state based on its local history of observations. However, the agents might come to different conclusions concerning that state, which complicates coordination between agents when selecting an optimal joint action. In fact, each agent has to anticipate the possible future behavior of all of its teammates in order to compute its own best action. This means in particular, that the notion of belief state has to be extended to include information about the other agents.

4.1.3 Solving DEC-POMDPs

Similar to what has been defined in the context of single-agent decision problems, we state the multi-agent optimization problem as finding a joint policy that maximizes the expected amount of accumulated reward when each policy is executed by the corresponding agent. We set $V(s_0, \boldsymbol{q})$ as the *expected value* of executing joint policy $\boldsymbol{q}$ from start state s_0, which means the expected sum of rewards collected by the agents when executing $\boldsymbol{q}$. This value can easily be calculated using the model parameters $\mathcal{T}$ and $\mathcal{O}$ of the DEC-POMDP. Solving a DEC-POMDP thus translates to identifying a joint policy that satisfies the following equation :

$$\boldsymbol{q}^* = \arg\max_{\boldsymbol{q} \in \boldsymbol{Q}} V(s_0, \boldsymbol{q}) \tag{4.1}$$

Each such joint policy $\boldsymbol{q}^*$ constitutes an optimal policy and thus a solution to our problem.

4.1.4 Multi-agent Belief States

Belief states capture the entire history of a partially observable Markov process, and are sufficient to select optimal actions. In a single-agent context, belief states are distributions over states. In a multi-agent environment, belief states also include a probability distribution over the possible future policies of the remaining agents [NTY+03, HBZ04]. If Q_i denotes the set of potential policies for agent i, and $\boldsymbol{Q_{-i}} = \langle Q_1, \ldots, Q_{i-1}, Q_{i+1}, \ldots, Q_n \rangle$ denotes the sets of policies for all other agents but agent i, then a multi-agent belief state for agent i is a distribution over $\mathcal{S}$ and $\boldsymbol{Q_{-i}}$:

Definition 4.1.3 (Multi-agent Belief State). *A* multi-agent belief state $\boldsymbol{b}_i$ *for agent* i *is a probability distribution over underlying states and future policies of all other agents :* $\boldsymbol{b}_i \in \Delta(\mathcal{S} \times \boldsymbol{Q_{-i}})$.

Contrary to single-agent belief states, which are of constant dimensionality, namely the dimensionality of the state space, multi-agent belief states are of variable dimensionality : the belief over some other agent's policies effectively depends on the policy space of that agent. It should also be noted that the multi-agent belief state reduces to a single-agent belief state in the case of a single agent DEC-POMDPs.

4.1.5 Multi-agent Value Functions

The multi-agent belief state synthesizes an agent's local partial view of the multi-agent environment. It permits the definition of a *local value function*, defined over the agent's local belief space. If we denote V_i the value function for agent i, $\boldsymbol{q}_{-i}$ the joint policy for all other agents but agent i, and $\langle \boldsymbol{q}_{-i}, q_i \rangle$ a complete joint policy, then the value of policy q_i in belief state $\boldsymbol{b}_i$ can be written as follows :

$$V_i(\boldsymbol{b}_i, q_i) = \sum_{s \in \mathcal{S}} \sum_{\boldsymbol{q}_{-i} \in \boldsymbol{Q_{-i}}} \boldsymbol{b}_i(s, \boldsymbol{q}_{-i}) V(s, \langle \boldsymbol{q}_{-i}, q_i \rangle) \tag{4.2}$$

It is important to note that a local policy cannot be evaluated by itself and without taking into account the dynamics of the other agents. This is true for competitive as well as for cooperative settings. In this sense, the decentralized control problem can be interpreted as a special case of a cooperative game. In game theory, the notion of *best response* captures the fact that a policy is the best choice, given that all other players have already made their choice.

Definition 4.1.4 (Best Response Policy). *We say that* $BR_i(\boldsymbol{b}_i, Q_i)$ *is the* best response policy *among the set of candidate policies* Q_i *of agent* i *and for belief state* $\boldsymbol{b}_i$ *:*

$$BR_i(\boldsymbol{b}_i, Q_i) = \arg\max_{q_i \in Q_i} V_i(\boldsymbol{b}_i, q_i)$$

The best response policy is the policy that optimally completes the behavior of the group, given that all agents $j \neq i$ already know which policy they should execute. Determining agent j's optimal policy however requires that all agents - and thus agent i - have made a decision about their optimal policy. This obviously leads to circular dependencies, and determining the optimal joint policy through n individual optimizations is therefore not possible. We will see that the dynamic programming approach proceeds the other way around : instead of determining the best response policies, it removes all those policies that can never constitute a best response in any

case. A policy is said to be *useful* if it constitutes a best response for at least one belief state. Otherwise, the policy is said to be *dominated*.

Definition 4.1.5 (Dominated Policy). *A policy $q_i \in Q_i$ is said to be* dominated *if it is a suboptimal policy over the entire belief space, which means that for each belief state $\boldsymbol{b}_i$, there is at least one other policy $\tilde{q}_i$ that is at least as efficient :*

$$(\forall \boldsymbol{b}_i)(\exists \tilde{q}_i \in Q_i \backslash \{q_i\}) \quad \textit{such that} \quad V_i(\boldsymbol{b}_i, \tilde{q}_i) \geq V_i(\boldsymbol{b}_i, q_i)$$

A policy that is not completely dominated is a useful policy.

4.1.6 Examples

Many decentralized real world problems can be captured by the decentralized Markov decision process framework, and we briefly mentioned some of them in the introduction. We will give a more detailed description of some DEC-POMDP problems in Chapter 9.

4.2 The MTDP Model

The *multi-agent team decision problem*, introduced by Pynadath and Tambe [PT02], is an alternative discrete Markov framework for decentralized cooperative control problems, very similar to the DEC-POMDP formalism. It extends in particular earlier work on team decision problems established by Ho [Ho80]. We will basically present it as an alternative to the DEC-POMDP, knowing that certain algorithms are based on it.

Definition 4.2.1 (MTDP). *A* multi-agent team decision problem *MTDP is defined as a tuple $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O}, \boldsymbol{\mathcal{B}} \rangle$, where*

- *$\mathcal{S}$ is a finite set of states $s \in \mathcal{S}$,*
- *$\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is a finite set of joint actions $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$, and $\mathcal{A}_i$ represents the set of actions a_i of agent i,*
- *$\mathcal{T}(s, \boldsymbol{a}, s')$ is the probability of transitioning from state s to state s' after execution of joint action $\boldsymbol{a}$,*
- *$\mathcal{R}(s, \boldsymbol{a})$ is the reward produced by the system when transitioning from state s to state s' after execution of joint action $\boldsymbol{a}$,*
- *$\boldsymbol{\Omega} = \Omega_1 \times \cdots \times \Omega_n$ is a finite set of joint observations $\boldsymbol{o} \in \boldsymbol{\Omega}$, and Ω_i represents the set of observations o_i of agent i,*
- *$\mathcal{O}(s, \boldsymbol{a}, \boldsymbol{o}, s')$ is the probability of observing joint action $\boldsymbol{o}$ when the system transitions from state s to state s' after execution of joint action $\boldsymbol{a}$,*
- *$\boldsymbol{\mathcal{B}} = \mathcal{B}_1 \times \cdots \times \mathcal{B}_n$ is the set of combined belief states.*

The additional component of the MTDP is the explicit mention of an agent's belief space $\mathcal{B}_i$ within the definition of the framework. Such a belief can now be computed during execution via various methods, and the mapping of local histories of observations to beliefs can be specified in more detail through specific *state estimator functions*. The MTDP model thus enables a more restrictive definition of multi-agent belief states than the one given in Definition 4.1.3. These belief states may not be sufficient to compute "optimal" policies as required by the DEC-POMDP formalism, but since the definition of the belief space is now integrated into the definition of the MTDP framework, it is possible to compute policies that are optimal, *given belief space $\mathcal{B}$*.

The explicit consideration of beliefs leads the authors to require that policies are defined over belief states :

Definition 4.2.2 (Domain-level policy). *A* domain-level policy *is a mapping from belief states to actions* $\pi_i : \mathcal{B}_i \rightarrow \mathcal{A}_i$.

The question related to the MTDP model is whether an explicitly modeled belief space either permits to express a larger class of problems, or if it helps reducing the complexity of the algorithms to solve it. It has been shown in the context of POMDPs that a belief over system states is a sufficient information for solving POMDPs optimally, and that Bayesian updating of beliefs, based on actions and incoming observations, leads to a coherent view of the system. In a similar way, belief states for decentralized POMDPs can be defined that permit to solve decentralized control problems optimally, and we will address this topic in Section 6.2.2 in more detail. An explicit modeling of belief states does thus not seem to be necessary.

One of the reasons why solving decentralized control systems optimally is hard lies in the necessity of considering all possible past information of all participating agents. Sometimes however, treating this exhaustive amount of information is not possible, due to bounded memory for example. Some other times, controllers have to content themselves with simpler solutions methods that might imply to actively discard some information. The authors call *imperfect recall* the fact that the agent does not have access to all of its past observations, as opposed to *perfect recall*. In cases of imperfect recall, of bounded memory, or of limited computing power, an explicitly and adequately defined belief space might help simplifying the solution process. Seuken and Zilberstein could show that the MTDP model and the DEC-POMDP model are equivalent under the perfect recall assumption :

Theorem 4.2.1. *The MTDP model and the DEC-POMDP model are equivalent under the perfect recall assumption.*

Proof. See [SZ05]. □

4.3 The I-POMDP Model

Another possible extension of the POMDP framework to multiple decision makers is the I-POMDP model, introduced by Gmytrasiewicz and Doshi [GD05]. As the authors formulate it, their model is supposed to be a "*subjective counterpart to an objective external view*" of a multi-agent system. The model focuses particularly on the local belief state of each agent, and its interaction status with other agents, hence its name : *interactive POMDP*. It is inspired by single-agent POMDPs and game theory, and applies to cooperative as well as to non-cooperative settings.

The main concept of the I-POMDP, adopted from game theory, is that of a *model* of an agent. A model captures all local information that is not directly accessible by the other agents. This includes for example beliefs or decision functions. In game theory, such local information is also referred to as a *type*. The state space of the I-POMDP is expanded to include the current model of each participating agent, and belief states thus become beliefs over system states and models. We introduce the I-POMDP model because of its close relationship with our point-based dynamic programming algorithm for decentralized POMDPs, which we will present in Section 6.2.2.

The first notion we have to define is that of an agent's *model*. A model is a possible local configuration of the agent.

Definition 4.3.1 (Model). *A* local model *for agent i is defined as $m_i = \langle h_i, f_i \rangle$, where*
- *$h_i \in H_i$ is a local history of observations,*
- *$f_i : H_i \rightarrow \Delta A_i$ is a stochastic action selection function.*

Given a class of possible models M for each agent, the definition of the state space is then extended to include the current local model of each agent. This new state now accounts for the presence of other decision makers within the system.

Definition 4.3.2 (Interactive State). *An* interactive state is_i *for agent i includes a real system state and one local model for each other agent. The interactive state space IS_i for agent is thus defined as $IS_i = \mathcal{S} \times \boldsymbol{M_{-i}}$.*

These two definitions lead to the I-POMDP formalism, a local mathematical model that expresses the subjective point of view that each agent has about the system.

Definition 4.3.3 (I-POMDP). *An* interactive POMDP *for agent i is a tuple $\langle IS_i, \boldsymbol{\mathcal{A}}, \mathcal{T}, \Omega_i, \mathcal{O}_i, \mathcal{R}_i \rangle$, where*
- *IS_i is the interactive state space,*
- *$\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the set of joint actions,*
- *$\mathcal{T}(s, \boldsymbol{a}, s')$ is the state transition function,*
- *Ω_i is the local set of observations,*
- *$\mathcal{O}(s, \boldsymbol{a}, o_i, s')$ is the local observation function,*
- *$\mathcal{R}_i(is_i, \boldsymbol{a})$ is the local reward function.*

The definition of the I-POMDP resembles that of the DEC-POMDP, although it allows for individual reward functions and has slightly different parameters. We will now address the problem of solving an I-POMDP the way their authors describe it.

The main objective of the I-POMDP model is to apply computationally feasible resolution methods known from single-agent POMDPs, such as value iteration, to decentralized domains [GD05]. This means establishing a value function over an appropriate belief space. The *interactive belief space* is obviously a distribution over interactive states :

Definition 4.3.4 (Interactive belief state). *An* interactive belief state $\boldsymbol{ib_i}$ *is a distribution over interactive states : $\boldsymbol{ib}_i \in \Delta IS_i$.*

The authors of the I-POMDP model then consider a particular class of models, which they call *intentional models*, and which include explicit reasoning about the presence of other agents. Note that this is not required by the definition of the model : agents can have very limited policies based on no-information models. Intentional models explicitly represent the local belief of the other agents, for example their estimation of the underlying system state.

Since solving an I-POMDP leads to the definition of a value function over beliefs, the question of updating interactive belief states arises. As in single-agent POMDPs, a belief at time t is a function of the previous belief, the last action taken, and the new observation. However, two additional factors have to be taken into account : first, the transition function now depends on the joint action taken by all agents, which in turn depends on the local models. Second, the models themselves can change as a function of local beliefs and new observations. The belief update of some agent i thus includes the consideration of possible belief updates of all other agents - which in turn have to consider the belief update of agent i itself. Such circular "*I think*

that you think that I think" reasoning especially appears in synchronous systems, and it is the main barrier for solving I-POMDPs optimally.

The need for infinitely nested belief updates has led the authors to consider approximations based on finite reasoning. An interactive belief state can now have a certain *depth*, which means that it includes higher-order reasoning about other agents beliefs until a certain level. A level-0 belief state is essentially a single-agent POMDP belief state, which only considers underlying system states. A level-1 belief state also includes level-0 belief states of the other agents. A *finitely nested* I-POMDP then is a system where the belief reasoning of each agent is only pursued until a given level. Under this restriction, optimal policies can be computed.

Definition 4.3.5 (Finitely Nested I-POMDP). *A* finitely nested I-POMDP *of level l is an I-POMDP where belief updates are terminated after l steps of recursion.*

The advantage of the finitely nested I-POMDP is that classical properties of POMDP value functions, such as piecewise linearity and convexity, can be preserved [GD05]. At the same time, the general complexity of the I-POMDP has not yet been established. The authors presume that, given a bound m on the possible models, and a strategy level l, solving an I-POMDP reduces to solving $O(m^l)$ POMDPs, which would mean that they are PSPACE-hard for finite-horizon problems, and undecidable for infinite-horizon problems. This statement however is not based on a formal analysis, and other authors suppose that the I-POMDP belongs to a higher complexity class [SZ05].

The I-POMDP model leads to different solution methods than those of the DEC-POMDP model, and it is so far an open question whether they are able to outperform the approximative algorithms known for DEC-POMDPs, and whether their complexity can be guaranteed to be lower, at least for a certain class of problems.

4.4 The Interac-DEC-MDP Model

The I-POMDP model focuses on the local view that each agent might have about the global state of the system. Thomas proposed in his PhD thesis to emphasize much more on the global aspects, namely the interaction between agents. Inspired by biological experiments that were conducted on rats, he developed a formal model, the DEC-MDP with interactions, that could reproduce some basic animal behaviors. He defines the *Interac-DEC-MDP* as a DEC-MDP with an additional interaction module [Tho05]. According to his definition, an interaction I is a *proposition* made by an agent to a group for accomplishing a certain task together. This proposition can be accepted, declined, or modified by the group, and the result of the interaction is the agreement on one of several possible joint actions - which will then be executed by the group - of a predefined set of *results* RI. A stochastic transition function $TRI : \mathcal{S} \times RI \rightarrow \mathcal{S}$ finally solves the joint action and the system transitions to a new state.

Executing an Interac-DEC-MDP means alternating the execution of two modules : the action module and the interaction module. In the action module, actions are selected according to a local policy, based on local knowledge only. The system then transitions according to the transition function of the DEC-MDP. In the interaction module, agents have the opportunity to engage in an act of collective problem solving. Each agent may suggest the execution of an interaction, and this proposition then has to be resolved collectively. An example for such an interaction could

be the exchange of an object between two agents, or the simultaneous execution of a joint action that requires the participation of several agents.

Learning or planning policies for the Interac-DEC-MDP is not obvious, since two separate policies, one for the action module, and one for the interaction module, have to be established, while their effects are heavily correlated. Evaluating the effect of a local action for example might depend on the success of the following interaction. Evaluating the usefulness of an interaction requires reasoning about the local policies that follow its execution. In his PhD thesis, Thomas proposes a learning algorithm based on Q-learning in order to compute the value functions for the action module and the interaction module. We believe that learning these two policies separately strongly resembles the alternation maximization principle from game theory (see Section 5.2). We therefore suspect the resulting policies to be in some form of an equilibrium.

The Interac-DEC-MDP model is not more expressive than the DEC-MDP model. It is so far rather restrictive in the way the interactions have to be engaged (strict alternation between the execution of the action and the interaction module), and no convergence guarantees have so far been established. It is however a valuable attempt to separate the mono-agent aspects of the problem solving process from the collaborative ones. In addition, it makes the important connection between decentralized decision theory, and collaborative, biology inspired, problem solving mechanisms and the theory of artificial animals, so called *animats* [Sig04].

4.5 Communication

In our description so far, a very important aspect of multi-agent systems seems not have been taken into account, namely inter-agent communication. It is obvious that the exchange of information is crucial to optimize cooperative behaviors, or to gain a sufficient degree of coordination. However, communication as a special case of action can easily be expressed by the DEC-POMDP formalism and is thus already implicitly included in the model.

4.5.1 The DEC-POMDP-Com Model

In order to show the expressiveness of the DEC-POMDP model, we introduce and extension that models communication explicitly, namely the *DEC-POMDP-Com framework*, introduced by [GZ03].

Definition 4.5.1 (DEC-POMDP-Com). *A* DEC-POMDP-Com *is given as a tuple* $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \Sigma, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$*, where*

- $\mathcal{S}$ *is a finite set of states* $s \in \mathcal{S}$*,*
- $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *is a finite set of joint actions* $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$*, and* $\mathcal{A}_i$ *defines the set of local actions* a_i *for agent* i*,*
- $\mathcal{T}(s, \boldsymbol{a}, s')$ *is the probability of transitioning from state* s *to state* s' *after execution of joint action* $\boldsymbol{a}$*,*
- Σ *is a finite set of messages* $\sigma \in \Sigma$*, and* $\boldsymbol{\sigma}$ *denotes a joint message,*
- $\mathcal{R}(s, \boldsymbol{a}, \boldsymbol{\sigma})$ *is the reward produced by the system when executing joint action* $\boldsymbol{a}$ *and sending joint message* $\boldsymbol{\sigma}$ *in state* s*,*
- $\boldsymbol{\Omega} = \Omega_1 \times \cdots \times \Omega_n$ *is a finite set of joint observations* $\boldsymbol{o} \in \boldsymbol{\Omega}$*, and* Ω_i *defines the set of local observations* o_i *for agent* i*,*

– $\mathcal{O}(s, \boldsymbol{a}, \boldsymbol{o}, s')$ *is the probability of jointly observing* $\boldsymbol{o}$ *when joint action* $\boldsymbol{a}$ *is executed in state* s.

According to the authors, solving a DEC-POMDP-Com means computing two local policies per agent, a *policy of action* and a *policy of communication*, such that their execution maximizes the expected sum of rewards accumulated by the system.

Definition 4.5.2 (Local policy of action). *A* local policy of action δ_i^A *is a mapping from local histories of observations and messages to actions :* $\delta_i^A : \Omega_i^* \times \Sigma^* \rightarrow \mathcal{A}_i$.

Definition 4.5.3 (Local policy of communication). *A* local policy of communication δ_i^Σ *is a mapping from local histories of observations and messages to messages :* $\delta_i^\Sigma : \Omega_i^* o_i \times \Sigma^* \rightarrow \Sigma$.

The tuple $\delta_i = \langle \delta_i^A, \delta_i^\Sigma \rangle$, constituted of a policy of action and a policy of communication, defines the local policy of an agent. Its execution is an alternate selection of an action to execute, and a message to send. This means that the system is always in an action or a communication phase, which is why the policy of communication also depends on the last observation o_i perceived after the execution of the policy of action.

We now show informally that explicit communication of messages can easily be integrated into the DEC-POMDP formalism, and this even in a much more general way than required by the above DEC-POMDP-Com framework. Suppose we wanted to extend a DEC-POMDP and add the possibility for agent i to send a single message symbol σ to a particular agent. This would mean adding $(n-1)$ new actions to the action set, namely sending the symbol σ to one of the remaining agents j. We will denote such an action $a_{i,\sigma,j}$. Each local observation $o_j \in \Omega_j$ is then replaced with two new observations $o_{j,\sigma}$ and $o_{j,\emptyset}$, indicating whether the message has been received or not. It remains adapting the transition, the observation, and the reward functions in order to model the reliability and the cost associated with sending messages.

Theorem 4.5.1 (Equivalence of DEC-POMDP and DEC-POMDP-Com). *The DEC-POMDP model and the DEC-POMDP-Com model are equivalent.*

Proof. It is obvious that a DEC-POMDP-Com can model a simple DEC-POMDP by keeping an empty message set. For simulating a DEC-POMDP-Com with a DEC-POMDP, one has to create an augmented process, similar to the way sketched above. One finally has to prove that the problem description does not increase by a factor more than polynomial compared to the original description. A complete proof can be found in [SZ05]. □

Although the DEC-POMDP model accounts for communication between agents, addressing the aspect of communication explicitly has several advantages, and helps modeling various research problems in multi-agent systems. One could ask for example :

– What is the optimal policy of action, given a policy of communication (and vice versa) ?
– What is the "meaning" of a message, what is the language ?
– What is the amount of information contained in a message ?

The question of optimal communication policies for different types of DEC-POMDPs has been partially addressed by Goldman [GZ03]. The language problem, namely the question of learning and translating the "meaning" of a message, has been established in [GAZ04]. We believe that the third point is particularly interesting, since it could lead to some new insights into the notion of *value of information* associated with a message, although it does not seem to have been addressed by the community yet.

4.5.2 Value of Information

Determining the value of an information has been an outstanding challenge in artificial intelligence and information theory since the groundbreaking works of Shannon [Sha48]. We believe that control theory based on Markov decision processes could shed some new light on the field, although not much work has been done so far. The general idea is very simple : exploit the value function of a POMDP/DEC-POMDP in order to establish a relationship between the sending or receiving of a message, and the associated gain of reward. An illustrative example is that of an *oracle POMDP*, which is a single-agent POMDP that is connected to an omniscient oracle. Whenever the POMDP controller is uncertain about the underlying system state, it could ask the oracle to unveil the real state. The immediate value of the information VI associated to this query is the expected gain in reward :

$$VI(\boldsymbol{b}) = \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V(s) - V(\boldsymbol{b}) \tag{4.3}$$

If questioning the oracle is associated with a certain cost C, then it is beneficial to ask if the expected gain in reward exceeds the costs : $VI(\boldsymbol{b}) > C$. A visual interpretation of this fact can be obtained by analyzing the value function as shown in figure 4.1. Executing actions in a system

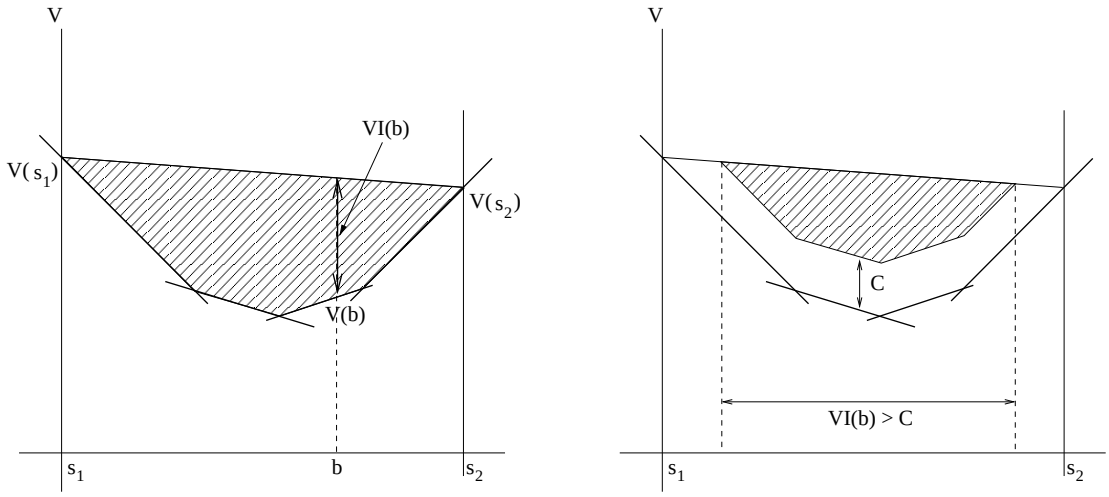

FIG. 4.1 – On the left : the shaded area indicates the value of additional information about the underlying state. - On the right : if the additional information has a certain cost C, then a region of the belief space can be identified where purchasing this additional information is expected to be beneficial.

with full and free access to its underlying state is sometimes called closed-loop control. If sensing incurs a cost, it can be beneficial to take actions in open-loop mode while estimating the evolution of the underlying hidden state. Hansen et al. have addressed the problem of optimizing a sequence of open-loop and closed-loop control actions in the context of single-agent reinforcement learning [HBZ96]. They included sensing costs in the Q-learning algorithm and show how this special case of an oracle POMDP can be solved optimally.

In a multi-agent setting, the state-action value function could help expressing the precise value of information associated with the sending of a message. Remember that the state-action

value function $Q(s, a)$ captures the expected amount of reward when executing action a in state s and following an optimal policy thereafter. By extending this concept to DEC-POMDPs, the *state-communication value function* should capture the expected amount of reward when sending joint message $\boldsymbol{\sigma}$, given the histories of observations and actions for all agents, and following an optimal joint policy thereafter : $Q(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n, \boldsymbol{\sigma})$. The value of information of an atomic message σ_i for the system could then be determined as the expected loss of reward if the message σ_i is replaced by another message σ_j :

Definition 4.5.4 (Value of Information). *The* value of information *associated with the sending of an atomic message σ_i in the context of histories $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$ is given by :*

$$VI(\sigma_i|\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n) = \min_{\sigma_j \neq \sigma_i} \left[Q(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n, \boldsymbol{\sigma_i}) - Q(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n, \boldsymbol{\sigma_j}) \right]$$

We believe that expressing the value of the information contained in a message by the expected gain of rewards of a Markov decision process is a novel idea that is worth further investigation.

4.5.3 Other Markov Models for Multi-agent Communication

The DEC-POMDP-Com model is not the only formalism that allows to address multi-agent communication explicitly. As there exists an extension of the DEC-POMDP model to handle communication explicitly, there also exists such an extension for the MTDP : the Com-MTDP, introduced by Pynadath and Tambe [PT02]. We will not introduce this model in more detail, since it has been shown that the Com-MTDP is itself equivalent to the DEC-POMDP-Com model [SZ05]. More generally, we will not discuss algorithms for explicitly planning communication acts, since they can usually be integrated in the DEC-POMDP framework itself.

Communication, and particularly learning to communicate, is also an issue in multi-agent reinforcement learning [GM04, SC04c]. We will cover some aspects of this issue in Chapter 8.

4.6 Particular Classes of DEC-POMDPs

Not all problem settings require the full expressiveness of the general DEC-POMDP model. There are for example scenarios where the decision process is decentralized, but where fully observability of the environment can be guaranteed. One might also think about problems where agents spent most of the time accomplishing rather independent tasks before collecting their results together. All these special cases are worth analyzing, and some of them even exhibit special structure that facilitates the problem solving process.

4.6.1 Jointly Fully Observable Systems

We have seen that partial observability complicates the resolution of the DEC-POMDP because of possible different local beliefs about the state of the system and the future policies of the agents. In a similar way as the fully observable MDP can be considered as a special case of the more general POMDP, a *jointly fully observable* DEC-POMDP, also called *DEC-MDP*, can be defined as follows [BGIZ02] :

Definition 4.6.1 (DEC-MDP). *A* DEC-MDP *is defined by $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$, where*
- *$\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$ is a DEC-POMDP,*

– *there is* joint fully observability*, which means that the underlying state can be determined with certainty as soon as the joint observation is known : if* $\mathcal{O}(s, \boldsymbol{a}, \boldsymbol{o}, s') > 0$ *then* $P(s'|\boldsymbol{o}) = 1$

In a DEC-MDP, the state of the system can thus be inferred by the team as a whole at each time step, but this information is distributed during runtime and would first have to be gathered, through communication for example. A system where each agent alone has full knowledge about the underlying state is called a *multi-agent MDP* or MMDP [CB98] :

Definition 4.6.2 (MMDP). *A* MMDP *is defined by* $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R} \rangle$*, where*
– $\mathcal{S}$ *is a finite set of states* $s \in \mathcal{S}$*,*
– $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *is a finite set of joint actions* $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$*, and* $\mathcal{A}_i$ *defines the set of actions* a_i *available to agent* i*,*
– $\mathcal{T}(s, \boldsymbol{a}, s')$ *is the probability of transitioning to state* s' *when joint action* $\boldsymbol{a}$ *is executed in state* s*,*
– $\mathcal{R}(s, \boldsymbol{a})$ *is the reward produced by the system when joint action* $\boldsymbol{a}$ *is executed in state* s*.*

Planning optimal policies for a MMDP is essentially similar to solving a MDP. This is not true anymore in the context of on-line learning. In this case, even the knowledge of the underlying state by all agents is still not sufficient to coordinate the agents properly. As described by Boutilier et al. in [CB98] for a simple 2-agent system, if joint actions $\boldsymbol{a_1} = \langle a, a \rangle$ and $\boldsymbol{a_2} = \langle b, b \rangle$ are both globally optimal actions in a given state, observed by both agents, but if the first agent decides to execute action $\boldsymbol{a_1}$, and the second agent decides to execute action $\boldsymbol{a_2}$, the resulting joint action $\langle a, b \rangle$ might be very suboptimal.

4.6.2 Independence Assumptions

In a decentralized control problem, the effects of the agents' decisions can be more or less correlated. There might be cases where a weakly-coupled system can actually be entirely decomposed into several independent problems, which then reduces the problem complexity. It is thus useful to provide an insight into the independences that might exist between the actions, the observations, or the rewards of the agents. There exist for instance many real-world scenarios where the multi-agent state space can easily be described as a cartesian product of single-agent states, such as individual robot positions. Becker et al. formalized such a state space factorization in [BZLG04] :

Definition 4.6.3 (Factored DEC-POMDP). *An* n*-agent DEC-POMDP is said to be* factored *if the state space can be decomposed into* n *distinct sets such that* $\boldsymbol{\mathcal{S}} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$*.*

Factorizing the state space permits to express independences between agents. If for example the position of the robot is part of the state space, then the local action `Go-North` will in general not influence the position, and thus the local state, of the other agents. In such a case, the transition function can also be decomposed into several functions that all capture a different aspect of the global transition of the system. This independence of the transition function has been formalized by Becker et al. in [BZLG04] :

Definition 4.6.4 (A transition independent DEC-MDP). *A factored DEC-MDP* $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$ *is said to be* transition independent *if the transition function* $\mathcal{T}$ *can be decomposed into* n *functions* $\mathcal{T}_1, \ldots, \mathcal{T}_n$ *such that :*

$$\mathcal{T}(\langle s_1, \ldots, s_n \rangle, \langle a_1, \ldots, a_n \rangle, \langle s'_1, \ldots, s'_n \rangle) = \prod_i \mathcal{T}_i(s_i, a_i, s'_i)$$

In a very similar way, one defines independence between observations. An observation independent DEC-MDP, as defined by [BZLG04], is given as :

Definition 4.6.5 (An observation independent DEC-MDP). *A factored DEC-MDP* $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$ *is said to be* observation independent *if the observation function* $\mathcal{O}$ *can be decomposed into* n *functions* $\mathcal{O}_1, \ldots, \mathcal{O}_n$ *such that :*

$$\mathcal{O}(\langle s_1, \ldots, s_n \rangle, \langle a_1, \ldots, a_n \rangle, \langle o_1, \ldots, o_n \rangle, \langle s'_1, \ldots, s'_n \rangle) = \prod_i \mathcal{O}_i(s_i, a_i, o_i, s'_i)$$

Finally, there can also be independences between the rewards. This means that the reward function can be decomposed in a similar way as described above, leading to n local reward function, and the definition of a reward independent DEC-MDP as given in [BZLG04] :

Definition 4.6.6 (Reward independent DEC-MDP). *A factored DEC-MDP* $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$ *is said to be* reward independent *if the reward function* $\mathcal{R}$ *can be decomposed into* n *functions* $\mathcal{R}_1, \ldots, \mathcal{R}_n$*, and if there exists an aggregation function* f *such that :*

$$\mathcal{R}(\langle s_1, \ldots, s_n \rangle, \langle a_1, \ldots, a_n \rangle) = f(\mathcal{R}_1(s_1, a_1), \ldots, \mathcal{R}_n(s_n, a_n))$$

A possible aggregation function is for example the sum $f(\mathcal{R}_1, \ldots, \mathcal{R}_n) = \sum_i \mathcal{R}_i$.

Identifying a possible reward independence between agents can be interpreted as the multi-agent version of the classical *credit-assignment problem* [Min61, SB98]. The credit-assignment problem concerns the definition of an appropriate reward function in the context of MDPs and POMDPs, namely the question which states, actions, or state-actions are beneficial for the solving of a given problem. This is not at all an obvious question and may need a lot of reasoning by the designer of an artificial system. The multi-agent version of the credit-assignment problem concerns the question how to *distribute* a common reward reasonably to the agents of a team. We will not address this issue furthermore, and always assume that the reward is given, but we define the multi-agent credit-assignment problem as follows :

Definition 4.6.7 (Multi-agent Credit Assignment Problem). *The* multi-agent credit assignment problem *is the problem of identifying whether a DEC-MDP is reward independent or not.*

The COIN framework for *collective intelligence*, proposed by Tumer and Wolpert, addresses this problem in the following way : given a global goal and a team of agents, the question is how the common reward function can be divided into local reward functions for each one of the agents, such that the initial goal can still be achieved. Each agent can then run a local optimization algorithm, which consists in simple reinforcement learning in the case of COIN [WWT99, TW00]. We argue that solving the COIN problem is equivalent to the multi-agent credit assignment problem, or the problem of identifying reward independence in DEC-POMDPs.

There are problem settings where the global state might not be observable, but where some local information is known with certainty. Consider for example a grid problem with several robots, such that the global state is given by the positions of the agents. Each agent might know its own grid position with certainty (or even its real position, using a GPS system for example), but has no knowledge about the position of the other robots. Such a setting is said to be *locally fully observable.*

Definition 4.6.8. *A factored DEC-POMDP is said to be* locally fully observable*, if, whenever an observation probability is non-zero* $\mathcal{O}(\langle s_1, \ldots, s_n\rangle, \langle a_1, \ldots, a_n\rangle, \langle o_1, \ldots, o_n\rangle, \langle s'_1, \ldots, s'_n\rangle) > 0$*, it is guaranteed that* $P(s'_i|o_i) = 1, (\forall i)$.

A very interesting property is that DEC-MDPs with independent transitions and independent observations are locally fully observable. This means that an optimal policy in this case is history independent.

Theorem 4.6.1. *A DEC-MDP with independent transitions and independent observations is locally fully observable. Its optimal policies are memoryless, and can be written for each agent as* $\pi_i : \mathcal{S}_i \rightarrow \mathcal{A}_i$.

Proof. See [GZ04]. □

4.6.3 Dependence Assumptions

Not only independence assumptions can facilitate the resolution of a control problem ; the same is also true for dependencies between variables, or correlations between the behaviors of the agents. Suppose for example a scenario where two robots are ready to pass a bridge, but where only one of them is allowed to be on the bridge at a time. If the first robot is on the bridge, then the action set of the second robot can be considered to be constrained. More generally, there exist scenarios where the set of available actions of an agent depends on the global state. They have recently been studied by Guo and Lesser in [GL05] and are based on the notion of *state-dependent action sets.*

Definition 4.6.9 (State-dependent Action Set). *Given a DEC-POMDP, the* state-dependent action set *for agent i is determined by a mapping* $\mathcal{B}_i : \mathcal{S} \rightarrow 2^{\mathcal{A}_i}$*, such that each state* $s \in \mathcal{S}$ *is mapped to a subset of* $\mathcal{A}_i$.

For the above example, where the global state could be a cross-product of the two local states *on-land* and *on-bridge*, such a mapping could be defined as follows : $\mathcal{B}_1$(*on-land*,*on-land*) = (*get-on-bridge*), $\mathcal{B}_1$(*on-land*,*on-bridge*) = $\emptyset$, $\mathcal{B}_2$ (*on-land*,*on-land*) = (*get-on-bridge*), and $\mathcal{B}_2$(*on-bridge*,*on-land*) = $\emptyset$.

Decentralized control problems with state-dependent action sets can sometimes lead to computationally lighter algorithms and approximations. An idea presented in [GL05] works by first considering for each agent the optimistic and the pessimistic action sets. These action sets lead to upper and lower bounds on the value function, and actions whose upper bound estimate is lower than some other actions lower bound estimate can finally be eliminated. The identification of interval-dominated actions has been extensively studied by Kaelbling [Kae93], and the elimination of dominated policies in decentralized Markov problems is a process that will be further discussed in Chapter 6.

4.6.4 Homogeneity Assumptions

Exploiting particular structure in the transition, observation, or reward function is one way of characterizing simpler classes of multi-agent systems. Another way is by restricting the agent design. Especially if agents are very small and simple entities that do not differ much one from another, the associated decision problem can possibly be stated in a different way. We begin by introducing *symmetric* DEC-POMDPs, namely systems where agents are physically interchangeable.

Definition 4.6.10 (Symmetric DEC-POMDP). *We call a factored DEC-POMDP* $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$ symmetric*, if all agents and their effects are interchangeable. This means that, for all two agents* i *and* j*, and all two joint actions* $\boldsymbol{a^1} = \langle a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n \rangle$ *and* $\boldsymbol{a^2} = \langle a_1, \ldots, a_j, \ldots, a_i, \ldots, a_n \rangle$ *with* a_i*,* a_j *interchanged, there exist*

$$\boldsymbol{s^1} = \langle s_1, \ldots, s_i, \ldots, s_j, \ldots, s_n \rangle, \quad \boldsymbol{s^2} = \langle s_1, \ldots, s_j, \ldots, s_i, \ldots, s_n \rangle \quad \textit{with } s_i, \; s_j \textit{ interchanged}$$

and

$$\boldsymbol{o^1} = \langle o_1, \ldots, o_i, \ldots, o_j, \ldots, o_n \rangle, \quad \boldsymbol{o^2} = \langle o_1, \ldots, o_j, \ldots, o_i, \ldots, o_n \rangle \quad \textit{with } o_i, \; o_j \textit{ interchanged}$$

such that

$$\mathcal{T}(\boldsymbol{s}, \boldsymbol{a^1}, \boldsymbol{s^1}) = \mathcal{T}(\boldsymbol{s}, \boldsymbol{a^2}, \boldsymbol{s^2})$$

and

$$\boldsymbol{O}(\boldsymbol{s}, \boldsymbol{a^1}, \boldsymbol{o^1}) = \boldsymbol{O}(\boldsymbol{s}, \boldsymbol{a^2}, \boldsymbol{o^2}).$$

Symmetric DEC-POMDPs allow to express limitations in the agent design - their hardware - as they appear rather frequently in real-world scenarios. They are usually constituted of a single class of agents, as it is often the case in the RoboCup domain for example. Each agent however might follow a completely different policy. If we want to express the fact that agents are strictly identical, including their policy - their software - we have to introduce *uniform* DEC-POMDPs :

Definition 4.6.11 (Uniform DEC-POMDP). *We say that a DEC-POMDP is* uniform*, if all agents are strictly identical. This means in particular that they all follow the same policy. A uniform DEC-POMDP is by definition symmetric.*

The optimization criterion for uniform DEC-POMDPs thus translates to *finding a single policy that, if applied by all agents, maximizes the expected cumulative amount of reward* collected by the team. This is an important difference to general DEC-POMDPs. We will see later that certain algorithms, namely heuristic search, can be sped up if knowing that the DEC-POMDP is uniform, whereas others, namely dynamic programming based algorithms, cannot be applied anymore.

4.6.5 Task-oriented Frameworks

Some decentralized control problems are particularly task oriented, which means that the agents aim to accomplish a well defined result or reach one of several goal states, such as in robotic soccer. This may imply reasoning about possible temporal or precedence constraints among actions, and it is hoped that adequate algorithms which exploit such dependencies may permit solving larger robotic problems.

The simplest version of a task-based problem is that of a *goal-oriented* system. It is essentially a single-task problem where, at some deadline T, one of several possible global goal states has to be reached by the agents. If this is the case, a special reward is assigned. Goldman and Zilberstein define a goal-oriented DEC-MDP in [GZ04] as follows :

Definition 4.6.12 (GO-DEC-MDP). *A finite-horizon DEC-MDP is* goal-oriented*, if the following two conditions hold :*

1. *There exists a special subset of states* $\mathcal{G} \subseteq \mathcal{S}$ *which are global goal states*

2. *The global reward is decomposable* $\mathcal{R}(s, \boldsymbol{a}) = \sum_i \mathcal{R}_i(s, a_i)$*, and an additional joint reward* $\mathcal{JR}(s)$ *is given to the system if reaching a global goal state* $s \in \mathcal{G}$ *before the deadline given by the horizon.*

Solving a goal-oriented DEC-MDP means finding a set of policies that help the whole group to accomplish one of the goals defined by G. It does not imply that each agent has a local goal to achieve. Although the above definition only holds for finite horizons and single goals, one could imagine decomposing larger problems into a sequence of several simpler goal-oriented DEC-MDPs. The reason why it is hoped that solving goal-oriented DEC-MDP might be easier than solving highly nested problems is that, once a goal is fixed, each agent may pursue this goal independently. If the system verifies some independence assumptions about the agents transitions, then the complexity of solving goal-oriented DEC-MDP effectively drops down (see section 4.7).

Another approach for considering task-oriented multi-agent problems is to start from an entirely decentralized system of n independent MDPs, defined over a factored state space. Such is the approach proposed by Beynier and Mouaddib in [BM05, BM06]. Each agent is now given a *mission*, a set of *local tasks* whose stochastic accomplishment is expressed by MDPs. The individual missions are in general correlated, which means that agents have to help each other in accomplishing their tasks. Such dependencies are expressed by a mission graph. An illustrative example is that of planetary rovers, analyzing an unknown environment such as the planet Mars. At the beginning of each day, a mission control center sends a list of tasks to the agents, which then have to plan their execution. Some tasks may be constrained by the time of the day, such as taking pictures, and some other tasks might require the previous execution of a task by a different agent, such as the analysis of rock samples that have to be uncovered first.

Given a mission, which is a set of tasks $w \in \mathcal{W}$, the agent's decision includes reasoning about when to execute which task. This local decision problem can be formalized as a MDP, which leads to a problem description with one MDP per agent. Each agent basically solves its own MDP, but it also takes into account the possible loss of local rewards for all other agents when executing a certain task. This influence of local actions of agent i on the local rewards of another agent j is expressed by an additional term, the *opportunity cost*. The opportunity cost OC_{w_i,w_j} for agent i's task w_i related to agent j's task w_j is deduced from two points of agent j's value function, namely the value for executing task w_j right now, and the value for executing it after the delay due to the prior execution of task w_i by agent i :

$$OC_{w_i,w_j} = V_i^{w_j,0} - V_i^{w_j,\Delta(w_i)} \tag{4.4}$$

The opportunity cost measure helps decentralizing the solution process, in that it gives each agent the possibility to solve its own local problem. Together with the local MDPs, it constitutes the *opportunity cost DEC-MDP* :

Definition 4.6.13 (OC-DEC-MDP). *An* opportunity cost *DEC-MDP is given by a set of n local MDPs* $\langle \mathcal{S}_i, \mathcal{W}_i, \mathcal{T}_i, \mathcal{R}_i \rangle$*, where*

- $\mathcal{S}_i$ *is a set of local states,*
- $\mathcal{W}_i$ *is a set of tasks,*
- $\mathcal{T}_i(s, w, s')$ *is the state transition function,*
- $\mathcal{R}_i(s, w)$ *is the reward function.*

The action set is a cross product of the tasks and a starting time. The precedence constraints then limit the amount of possible starting times to a finite set.

A similar way of modeling task-oriented problems by explicitly formalizing the dependencies between tasks has been established by Becker et al. in [BZL04]. The authors focus particularly on the notion of *events*. An event is basically a state transition, and dependencies occur between events. In general, there are events that have to happen first in order for some transitions to succeed. These dependencies are the only interactions that are allowed between agents :

- a *primitive event* $e = (s, a, s')$ is equivalent to an atomic transition,
- an *event* $E = \{e_1, \ldots, e_k\}$ is a set of primitive events, and
- a *dependency* between agent i and agent j is $d = (E_i, D_j)$, where E_i is an event for agent i, and D_j is a set of state-action pairs for agent j.

The solution of multiple tasks by a team of agents can often be captured by the definition of the appropriate dependencies of events. Planning the local behavior for the section in between these events is basically a single-agent control problem, and Becker et al. show that the complexity of solving a DEC-MDP with *event-driven interactions* has complexity exponential in $|\mathcal{S}|$ and doubly exponential in the number of dependencies. Note that this is in general much better than the worst case complexity of DEC-MDPs without exploiting further structure. The reason why this class is easier to solve is because its policy space is reduced : only the history of interactions between agents has to be remembered, whereas all other information is only relevant on a local scale and can be separated from the true multi-agent part of the problem.

The task-oriented models presented above restrict the general expressiveness of the Markov framework, and require a priori knowledge about the system, but they translate some very specific real-world scenarios and may help simplifying the search for solutions.

4.7 Complexity Results

We have seen that fully observable MDPs are P-complete, and that partially observable MDPs are PSPACE-complete for the finite-horizon case. A longstanding question was whether decentralized systems could efficiently be reduced to their single-agent counterpart, or if they are harder to solve. A first step was done by Papadimitriou and Tsitsiklis, who showed that a simple decentralized control problem involving two agents is NP-hard [PT82, PT86]. Bernstein et al. then showed that solving a DEC-POMDP does provably not admit a polynomial time solution. Given a DEC-POMDP, a value K, and an initial state s_0, the question whether there exists a joint policy that yields an expected reward of at least K when being executed from s_0 needs an exponential time to be answered.

Theorem 4.7.1. *The finite-horizon DEC-POMDP is NEXP-complete. The discounted infinite-horizon DEC-POMDP is undecidable.*

Proof. See [BGIZ02]. □

Under the assumption that the complexity classes EXP and NEXP are distinct, a fact that is strongly believed is true, but that has not yet been proven, solving a DEC-POMDP thus needs superexponential time.

The question then occurred whether particular DEC-POMDPs admit simpler solutions. Since one of the reasons for the higher complexity of decentralized systems seems to be the multiplied

partial observability of the agents, one could hope that DEC-MDPs, where agents jointly fully observe the system state, admit easier solutions. Unfortunately, this is not true.

Theorem 4.7.2. *The finite-horizon DEC-MDP is NEXP-hard.*

Proof. See [BGIZ02]. □

The main reason why DEC-MDPs are not any easier to solve is that agents still need to remember the entire history of observations in order to make optimal decisions. This condition however can be relaxed when the system admits a factored state space with independent transitions and observations. In such a case, the evolution of the local state only depends on the agents own actions. It can even be guaranteed that each agent has always full access to its local state.

Lemma 4.7.1. *A factored DEC-MDP with independent transitions and observations is locally fully observable.*

Proof. See [GZ04]. □

The local full observability together with the independence properties leads to a partially decorrelated system, where the cohesion is only given by the common reward function. Such a system is indeed easier to solve.

Theorem 4.7.3. *The finite-horizon DEC-MDP with independent transitions and observations is NP-complete.*

Proof. See [GZ04]. □

Several other sub-classes of DEC-POMDPs are even easier to solve. Constraining the global reward to the reaching of a specific goal state leads to polynomial complexity. This implies restricting a GO-DEC-MDP to a single goal state, and supposes that the reward function be *uniform* : all actions incur the same cost.

Theorem 4.7.4. *The GO-DEC-MDP with independent transitions and observations, a single goal state, and uniform cost is P-complete.*

Proof. See [GZ04]. □

The above goal-oriented DEC-MDP is a rather degenerated decentralized control problem, and resembles much a stochastic shortest path planning problem with multiple decision makers. The same is true for the MMDP formalism, which is really an MDP with joint actions, and can thus be solved in polynomial time.

Recently, Shen et al. showed that the complexity of solving a decentralized MDP can be deduced from the complexity of the interactions between agents [SBL06]. This may be one reason for focusing on the interactions explicitly, as it has been done in the Interac-DEC-MDP model for example. It also explains why independence properties, such as independence of transitions, lead to a drop in complexity : if agents evolve independently of each other, their interaction history is necessarily smaller. The crucial point is whether the interaction history is *polynomially encodable*.

Theorem 4.7.5. *The DEC-MDP with a polynomial encodable interaction history is NP-complete. The DEC-MDP that does not admit a polynomially encodable interaction history is harder than NP.*

Proof. See [SBL06]. □

The authors are able to show that the interaction history of a DEC-MDP with independent transitions and observations is indeed polynomial encodable.

4.8 Conclusion

We introduced several extensions of the POMDP formalism to multiple cooperative decision makers, including the DEC-POMDP, the MTDP, the DEC-POMDP-Com, or the Com-MTDP model. Most of these models are equivalent, which is why we focus on one of them, namely the DEC-POMDP formalism, which we believe to be the most logical extension of POMDP theory to multi-agent problems. We have shown that the DEC-POMDP includes communication as a particular aspect of acting and observing the environment, and we have emphasized on describing several subclasses of the general DEC-POMDP, which allow simplifying the solution process by exploiting particular structure of the system. The worst case complexity of the DEC-POMDP has been shown to be more then exponential in the problem size, which means that solving larger problems optimally is almost infeasible. However, the formalism is in important step forward in understanding decentralized control theory, and in developing smarter approximation techniques in the future.

Chapitre 5

Solving DEC-POMDPs using Game Theory

5.1 Introduction to Game Theory

The DEC-POMDP models decision problems associated with a team of cooperative agents. Game theory is the analogous discipline for modeling competitive scenarios. Surprisingly, it is much older than the theory of decentralized MDPs, and was established as a research field of its own by John von Neumann in the 40s. Several solution concepts from game theory have since been adopted to solving cooperative multi-agent decision problems, and this is one reason why we briefly introduce its general concepts in this chapter.

5.1.1 Matrix Games and Bayesian Games

A game is a collection of *players*, a set of *actions*, and a *payoff matrix*, which determines the individual payoff for each player, given a joint action. Such a game is rock-paper-scissor, which involves two players, three actions, and the payoff matrix given in Figure 5.1. The matrix form of

	rock	**paper**	**scissor**
rock	draw	A wins	B wins
paper	B wins	draw	A wins
scissor	A wins	B wins	draw

FIG. 5.1 – The game rock-paper-scissor.

a game is usually called its *normal form*. The normal form representation allows to make simple statements about the game. For the game shown in Figure 5.1, one can for example affirm that, if both players chose the same action, whatever that action may be, then the result of the game will always be a draw. It is also possible to answer questions such as : "*If my opponent plays* `paper`*, then what should I play ?*" (Answer : `scissor`). Matrix games are the most simple representation of a game. They do not allow to make any assumption about the players themselves. Sometimes however, it is essential to model the private information each player may have about the game or its opponents. This private information may influence the selection of actions, and thus the utility estimation of each action. In the game theory literature, the private information held by

an agent is called its *type*. Extending matrix games to games with private information leads to *Bayesian games*. A Bayesian game is a matrix game with additional type profiles for each agent.

Definition 5.1.1 (Bayesian Game). *A* Bayesian game *is given by a tuple* $\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\Theta}, \boldsymbol{\mathcal{R}} \rangle$*, where*
- $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *is a finite set of joint actions* $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$*, and* $\mathcal{A}_i$ *defines the set of actions* a_i *for player* i*,*
- $\boldsymbol{\Theta} = \Theta_1 \times \cdots \times \Theta_n$ *is a finite set of joint types* $\boldsymbol{\theta} \in \boldsymbol{\Theta}$*, and* Θ_i *defines the set of types* θ_i *for player* i*,*
- $\boldsymbol{\mathcal{R}}$ *is a joint reward matrix.*

Playing Bayesian games includes keeping a probability distribution over types, and updating this distribution after each play via Bayes' rule (hence the name of Bayesian game). At the beginning of the game, a prior distribution p over the initial types is distributed to all players.

5.1.2 Markov Games and POSGs

Rock-paper-scissor is a *single-stage* game, meaning that the game is over once all action choices have been executed. Games that unfold over several stages include an additional transition function, defined over an appropriate state set. They are called *stochastic games* or *Markov games*.

Definition 5.1.2 (Markov game). *A* Markov game *is a multi-stage game given by a tuple* $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\mathcal{R}} \rangle$*, where*
- $\mathcal{S}$ *is a finite set of states* $s \in \mathcal{S}$*,*
- $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *is a finite set of joint actions* $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$*, and* $\mathcal{A}_i$ *defines the set of actions* a_i *for agent* i*,*
- $\mathcal{T}(s, \boldsymbol{a}, s')$ *is the probability of transitioning from state* s *to state* s' *after execution of joint action* $\boldsymbol{a}$*,*
- $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$ *is a set of payoff functions, and* $\mathcal{R}_i(s, \boldsymbol{a})$ *defines the payoff given to agent* i *when joint action* $\boldsymbol{a}$ *is played in state* s*.*

The definition of the Markov game strongly resembles the definition of the MMDP (cf. Definition 4.6.2). The only difference is that each agent now possesses its own reward function, and thus tries to maximize its individual payoffs only. As it is the case for MDPs, the behavior of each player can be modeled by a decision function, its *strategy*. One commonly distinguishes deterministic from stochastic strategies.

Definition 5.1.3 (Pure strategy). *A* pure strategy π_i *for player* i *is a deterministic local policy that associates an action with each state* $\pi_i : \mathcal{S} \to \mathcal{A}_i$*. We call* strategy vector $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_n \rangle$ *a set of* n *strategies such that player* i *holds strategy* π_i*.*

Definition 5.1.4 (Mixed strategy). *A* mixed strategy π_i *for player* i *is a stochastic local policy that associates a distribution of actions with each state* $\pi_i : \mathcal{S} \to \Delta\mathcal{A}_i$*. We call* mixed strategy vector $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_n \rangle$ *a set of* n *mixed strategies such that player* i *holds strategy* π_i*.*

As for cooperative scenarios, one of the challenges in game theory is to compute a set of optimal strategies for all players. Defining optimality in the case of a game however is by far less obvious than it first might appear. It actually constitutes one of the major research efforts in game theory. If we take the above game of rock-paper-scissors, which joint strategy is the optimal one ? Should A win over B, or is a draw a better solution ? Obviously, determining the usefulness of an action of a single player is not always possible : the `rock` action for example may lead to all three possible outcomes, depending on the strategy of the opponent.

A very simple algorithm to learn a game is *fictitious play* [Bro51]. It consists in estimating the opponent's mixed strategy by counting the number of times it has played each one of its actions so far. The agent then choses to play the action that is optimal with respect to its current estimate of the opponent's mixed strategy. Unfortunately however, fictitious play does not necessarily converge to a stable solution. A different concept for solving games is needed.

5.1.3 The Nash Equilibrium

The difficulty in defining a global optimality criterion for competitive games has led John Nash to consider an alternative approach. Given a joint strategy, one can ask if all players are satisfied with the outcome of the game, or if a player would have selected another strategy given the actual strategies of the remaining players. For the game shown above, the joint action $\langle \texttt{rock}, \texttt{scissor} \rangle$ would have led agent B to have a regret about its action, since the `paper` action would have been a better choice. A game is said to admit an *Nash equilibrium* solution if no player has the local incentive to deviate from its current strategy.

Definition 5.1.5 (Nash Equilibrium). *A strategy vector $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_n \rangle$ is said to constitute a* Nash equilibrium *for the Markov game $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\mathcal{R}} \rangle$ if for each player i*

$$V_i(s, \langle \pi_i, \boldsymbol{\pi_{-i}} \rangle) \geq V_i(s, \langle \pi'_i, \boldsymbol{\pi_{-i}} \rangle), \qquad (\forall s \in \mathcal{S})(\forall \pi'_i)$$

where V_i is the local value function of player i. It can be determined as in MDPs by maximizing the payoffs obtained from reward function $\mathcal{R}_i$.

Nash has shown that the notion of equilibrium replaces the notion of optimality in the case of non-cooperative games. He also proved that each game admits at least one Nash equilibrium. For the example given above, the joint action $\langle \texttt{rock}, \texttt{scissor} \rangle$ clearly does not constitute a Nash equilibrium. In fact, no deterministic joint strategy constitutes an equilibrium, since at least one of the two players would always be willing to change its strategy choice. The equilibrium appears when considering mixed strategies, and consists in playing each action with probability 1/3.

Theorem 5.1.1 (Existence of a Nash Equilibrium). *Each Markov game with mixed strategies admits at least one Nash equilibrium.*

Proof. See [Nas51]. □

While the existence of the Nash equilibrium can be guaranteed, the same does not hold for its uniqueness. If several possible Nash equilibria exist, there is an additional question of equilibrium selection, which usually requires the coordination by an external mediator. Answering the question whether a strategy choice constitutes an equilibrium translates to asking each player if it is willing to keep its strategy choice, knowing the strategies of the other players. This is captured by the concept of *best response* which we introduced earlier in the context of DEC-POMDPs (see Section 4.1.5) : a joint strategy constitutes a Nash equilibrium if the strategy of each player is a best response to the strategies of the remaining players.

Definition 5.1.6 (Best Response Strategy). *A strategy $BR_i(s, \boldsymbol{\pi_{-i}})$ is said to be the* best response strategy *to strategy set $\boldsymbol{\pi_{-i}}$ in state s, if :*

$$BR_i(s, \boldsymbol{\pi_{-i}}) = \arg\max_{\pi_i \in \Pi_i} V_i(s, \langle \pi_i, \boldsymbol{\pi_{-i}} \rangle)$$

The alternative definition of the Nash equilibrium thus requires each player to hold a strategy that is the best response to the strategies held by the other players.

If the players do not have access to the global state of the game, the Markov game becomes partially observable. The *partially observable stochastic game* (POSG) is the partially observable extension of the Markov game as is the POMDP the partially observable extension of the MDP.

Definition 5.1.7 (POSG). *A* partially observable stochastic game *is given by a tuple* $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\mathcal{R}}, \boldsymbol{\Omega}, \mathcal{O} \rangle$*, where*

- $\mathcal{S}$ *is a finite set of states* $s \in \mathcal{S}$*,*
- $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *is a finite set of joint actions* $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$*, and* $\mathcal{A}_i$ *defines the set of actions* a_i *for agent* i*,*
- $\mathcal{T}(s, \boldsymbol{a}, s')$ *is the probability of transitioning from state* s *to state* s' *after execution of joint action* $\boldsymbol{a}$*,*
- $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$ *is a set of payoff functions, and* $\mathcal{R}_i(s, \boldsymbol{a})$ *defines the payoff given to agent* i *when joint action* $\boldsymbol{a}$ *is played in state* s*,*
- $\boldsymbol{\Omega} = \Omega_1 \times \cdots \times \Omega_n$ *is a finite set of joint observations* $\boldsymbol{o} \in \boldsymbol{\Omega}$*, and* Ω_i *defines the set of observations* o_i *of agent* i*,*
- $\mathcal{O}(s, \boldsymbol{a}, \boldsymbol{o}, s')$ *is the probability of joint observation* $\boldsymbol{o}$ *when joint action* $\boldsymbol{a}$ *is executed in state* s*, and the system transitions to state* s' *thereafter.*

The special case of a partially observable stochastic game where all agents share the same reward function is commonly called an *identical payoff* POSG. The definition is due to Peshkin et al. [PKMK00].

Definition 5.1.8 (POIPSG). *A* partially observable identical payoff stochastic game *is a POSG where all agents have the same reward function :*

$$\mathcal{R}_i = \mathcal{R}_j, \qquad (\forall i, j)$$

We state without proof that a POIPSG is equivalent to a DEC-POMDP. While the POIPSG is the cooperative variant of a partial observable stochastic game, the DEC-POMDP is the multi-agent version of a partially observable Markov decision process.

5.1.4 An Example : The Prisoners Dilemma

We now describe a famous game-theoretic example, a two-person game called *prisoners dilemma*, in order to illustrate the properties of game theoretic solutions in decentralized decision making. Two prisoners are held in a prison, each one in a separated cell. The police has insufficient evidence for their conviction, and thus offers each one of them the same deal : if one testifies against the other, and the other remains silent, then first one can leave the prison and the second one receives a full sentence of 10 years. If both testify against each other, they will both get sentenced for 2 years. If both stay silent however, the police can only keep them in prison for a minor charge and a stay of 6 months. This game has only a single state, and two actions per player, namely `betray` or `stay silent`. It can entirely be described by the matrix shown in Figure 5.2. The only Nash equilibrium of the game consists in both players choosing the `betray` action. In this case, both prisoners will get a sentence of 2 years. If however one of them changes its strategy and choses to stay silent, it will deteriorate its position and get a sentence of 10 years. No player thus has an incentive to deviate from this strategy. At the same time, cooperation of the players could lead to a much better scenario, where each prisoner choses to remain silent, in which case

	B remains silent	**Prisonnier B betrays**
A remains silent	(6 months, 6 months)	(10 years, liberated)
Prisonnier A betrays	(liberated, 10 years)	(2 years, 2 years)

FIG. 5.2 – The prisoners dilemma

both only get a minor sentence of 6 months. This solution however is not stable in a general competitive game, since each player would then have the unilateral incentive to deviate, to chose `betray` in order to be liberated immediately. As soon as the first prisoner betrays, the second one has also an incentive to betray, leading to the above mentioned stable equilibrium of the game. The Nash equilibrium of the prisoners dilemma is not globally optimal. It constitutes however a stable solution. This property should always be kept in mind when using game theoretic concepts for solving cooperative multi-agent decision problems.

5.1.5 The Tragedy of the Commons

A particular problem arises when competing self-interested agents act in a common environment without taking into account the interest of the group. There exist scenarios where selfish behavior actually deteriorates the benefit of the group in the long run. This is called the *tragedy of the commons*, and was first mentioned by William Foster Lloyd and later by Garrett Hardin [Har68]. The setting is simple : imagine a reward function that produces a positive local reward each time action `act` is being executed, but that also leads to a deterioration of the environment which affects all agents. If the local reward exceeds the penalty, then it is beneficial for a single agent to execute this action. If however all agents reason the same way, then the global deterioration may exceed the local benefit. However, if agents are purely self-interested, then there is no way of avoiding this "tragedy".

It is important to understand the tragedy of the commons when comparing solution concepts for competitive games and cooperative multi-agent systems. If game theoretic approaches, such as the alternation maximization principle, are used to solve cooperative multi-agent systems, then their inability to incorporate global reward dependencies might lead to poor overall performance, such as in the tragedy of the commons. In addition, these dependencies complicate the solution of such concepts as the COIN framework, or the decomposition of the reward function (see Section 4.6.2).

5.2 Solving Games

We assume that solving a game effectively means determining at least one of its Nash equilibria. For matrix games, Nash equilibria can be computed directly, using the Lemke-Howson method [CPS92] for example. For larger games however, or for cases where the solution process has to be decentralized, alternative techniques are required. One of them, the *alternation maximization principle*, has led to a variety of algorithms for cooperative games and decentralized Markov decision processes. Its underlying idea is very simple : since in a Nash equilibrium, each player has to play a best response strategy with respect to the strategies of the opponents, why not let each player compute its best response one after another until convergence is achieved ? Note that this process means reducing a multi-agent decision problem to its single-agent counterpart,

where a variety of algorithms with lower complexity can then be applied to. The alternation maximization principle, sketched in Algorithm 23, is the basis for all of the following techniques for solving cooperative multi-agent systems.

Algorithm 23 The alternation maximization algorithm - `AlternationMaximization()`

Require: A set of strategies $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_n \rangle$ and an evaluation function F
Ensure: A Nash equilibrium joint strategy $\boldsymbol{\pi}$
1: Initialize a random joint policy $\boldsymbol{\pi}$
2: **while** not converged **do**
3: Select an agent i at random
4: Compute π_i^*, the best response to $\boldsymbol{\pi_{-i}}$ under evaluation function F
5: Replace π_i with π_i^* in $\boldsymbol{\pi}$
6: **end while**

5.3 Solving DEC-POMDPs Using Game Theory

When we talk about solving DEC-POMDPs using methods from game theory, we mean identifying at least one Nash equilibrium by the use of the maximization principle. All of the subsequent algorithms will follow this idea.

5.3.1 Subjective Coevolution

An early work on solving multi-agent POMDPs by the alternation maximization is the *subjective coevolution algorithm* introduced by Chadès, Scherrer, Charpillet [CSC02, SC02]. Its underlying idea is to simplify the solution of a DEC-POMDP by constructing a set of simpler process models, and to solve these models for a common Nash equilibrium. The resulting algorithm has rather low complexity, which enables the solution of large multi-agent decision problems.

5.3.1.1 Subjective MDPs

A *subjective MDP* is an approximation of the local decision problem that each one of the agents of a multi-agent team is faced with. It obviously represents a simplification of the original process, but Chadès et al. motivate its definition by the inherent complexity of the DEC-POMDP model itself. A subjective MDP is basically an MDP that is based on observations instead of states, and whose transition function is then adapted accordingly.

Definition 5.3.1 (Subjective MDP). *Given a DEC-POMDP, a* subjective MDP *for agent i is a tuple $\langle \mathcal{O}_i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{R}_i \rangle$, where*
- *$\mathcal{O}_i$ is the finite set of observations defined for agent i,*
- *$\mathcal{A}_i$ is the finite set of actions defined for agent i,*
- *$\mathcal{T}_i : \mathcal{O}_i \times \mathcal{A}_i \to \mathcal{O}_i$ is the observation transition function for agent i,*
- *$\mathcal{R}_i : \mathcal{O}_i \times \mathcal{A}_i \to \mathbb{R}$ is agent i's local reward function.*

Solving a subjective MDP means finding a memoryless policy that deterministically maps the last observation to an action $\pi_i : \mathcal{O}_i \to \mathcal{A}_i$. We know from single-agent POMDP theory that such policies cannot be guaranteed to be optimal, but since the goal is to establish an approximative algorithm that enables to address larger multi-agent problems, its consideration can sometimes be justified.

The definition of a subjective MDP relies on several assumptions. The first one is that the observations for agent i capture a reasonable amount of local information on which it makes sense to define a transition and a reward function. An example is obviously a factored DEC-MDP, where the observations would correspond to local states (see Section 4.6.2). The second assumption is that the reward function is decomposable. In contrast to the property of reward independence encountered in DEC-MPDs, this includes defining an appropriate reward function based solely on local observations. The third assumption concerns the ability to define a coherent local transition function, also solely based on the agents local observations. This can be difficult, since the transition function has to capture the evolution of the system as a whole, while being limited to agent i's perception only. In general, these three assumptions are heavily correlated, and require a certain amount of a priori knowledge about the system in order to be fulfilled.

5.3.1.2 The Coevolution Algorithm

The coevolution algorithm is a simple application of the alternation maximization principle to a set of subjective MDPs. The idea is simple : fixing the policies for all agents but one, incorporating their behavior in the subjective MDP of that agent, and searching for a best response policy. Since policies are supposed to be deterministic, reactive, and memoryless, it is possible to establish a value function over observations, and to apply value iteration techniques to solve it. An optimal policy can then be extracted from the value function by one-step lookahead. Such an approach is summarized in Algorithm 24. The coevolution approach has successfully been applied to solve

Algorithm 24 The coevolution algorithm - `Coevolution()`

Require: A set of subjective MDPs $\langle \mathcal{O}_i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{R}_i \rangle_{1 \leq i \leq n}$
Ensure: A joint equilibrium policy $\boldsymbol{\pi}$
1: Initialize a random joint policy $\boldsymbol{\pi}$
2: **while** not converged **do**
3: Select an agent i at random
4: Determine and solve the subjective MDP$_i$ via value iteration
5: Compute $\pi_i^* = BR_i(o, \boldsymbol{\pi_{-i}})$, $(\forall o \in \mathcal{O}_i)$
6: Replace π_i with π_i^* in $\boldsymbol{\pi}$
7: **end while**

pursuit problems, where a team of agents has to hunt a prey on a grid. The observations were noisy indications about the position of the prey, and the positions of the teammates [CSC02]. It has been proven that the coevolution algorithm converges to a Nash equilibrium joint policy. It remains however an open question to which extend this equilibrium is related to the global Nash equilibrium of the system that one would expect to obtain by a more accurate modeling of the agents' behaviors.

5.3.2 JESP : Joint Equilibrium-based Search for Policies

An extension of the alternation-maximization principle to general DEC-POMDPs was presented by Nair et al. [NTY+03]. Instead of assuming reactive memoryless policies, the authors consider more complex policy classes such as policy trees, which increases the complexity of their algorithm, but allows for higher quality solutions. The underlying idea however is the same, and consists in generating an initial joint policy at random, and in alternately selecting one of the

agents to improve its local policy, until a global equilibrium is reached. This requires searching in the space of policy trees $q \in Q$. The joint equilibrium-based search algorithm (JESP) is shown in Algorithm 25. Since the number of possible policy trees of a finite depth is always finite, there

Algorithm 25 Exhaustive joint equilibrium-based search for policies - `Exhaustive-JESP()`

Require: A DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$, a horizon T, and a start state s_0
Ensure: A joint equilibrium policy $\boldsymbol{q}$
1: Initialize a random joint policy $\boldsymbol{q}$
2: **while** not converged **do**
3: **for all** agent i **do**
4: Compute $q_i^* = BR_i(s_0, \boldsymbol{q_{-i}})$ by searching the set of policy trees for agent i
5: Replace q_i with q_i^* in $\boldsymbol{q} = \langle q_1, \ldots, q_i, \ldots, q_n \rangle$
6: **end for**
7: **end while**

is only a finite set of possible joint policies, and the algorithm is guaranteed to terminate in an equilibrium [NTY+03]. However, computing the best response policy always implies considering the exhaustive set of horizon-T policies, hence the name of the algorithm.

We said earlier that fixing all policies but one reduces a multi-agent problem to a single-agent one. Instead of determining the best response policy by considering the exhaustive set of candidate policies over the entire horizon, several dynamic programming approaches for POMDPs can be applied that build such a policy incrementally. This is the general idea of the dynamic programming JESP algorithm. As stated earlier, multi-agent belief states have to take the presence of other agents into account. Nair et al. show how the fixed policies $\boldsymbol{q_{-i}}$ and the model parameters can be used to determine, for each horizon, the belief states that are needed to compute agent i's value function [NTY+03]. They also note that the individual value functions are piecewise linear and convex, thus potentially allowing continuous POMDP solution techniques to be applied. The dynamic programming JESP (DP-JESP) has obviously significant advantages in runtime compared to the exhaustive JESP, and it is sketched in Algorithm 26. Note that both the JESP and the DP-JESP algorithms are still restricted to converge to local optima, namely Nash equilibria.

Algorithm 26 Dynamic programming joint equilibrium-based search for policies - `DP-JESP()`

Require: A DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$, a horizon T, and a start state s_0
Ensure: A joint equilibrium policy $\boldsymbol{q}$
1: Initialize a random joint policy $\boldsymbol{q}$
2: **while** not converged **do**
3: **for all** agent i **do**
4: Fix all policies $\boldsymbol{\pi_{-i}}$ and compute V_i via dynamic programming
5: Extract q_i^* from V_i
6: Replace q_i with q_i^* in $\boldsymbol{q} = \langle q_1, \ldots, q_i, \ldots, q_n \rangle$
7: **end for**
8: **end while**

5.3.3 Bayesian Games Approximation

While the previous algorithms either did not model private beliefs, or determined belief sets during solution only, Emery-Montemerlo et al. suggested to include the private information held by each agent explicitly in the definition of the games themselves [EMGST04]. Their idea is the following : model the entire history of the decision process until some time t as a single entity, which is then captured by the types θ_i^t of the agents at time t, and determine the optimal next action by one-step lookahead. This means translating the decision problem at time t into a Bayesian game, and solving this game for an equilibrium. A schematic representation of the Bayesian game approximation, which can take a DEC-POMDP, a POSG, or any other decentralized model as an input, is given in Figure 5.3.

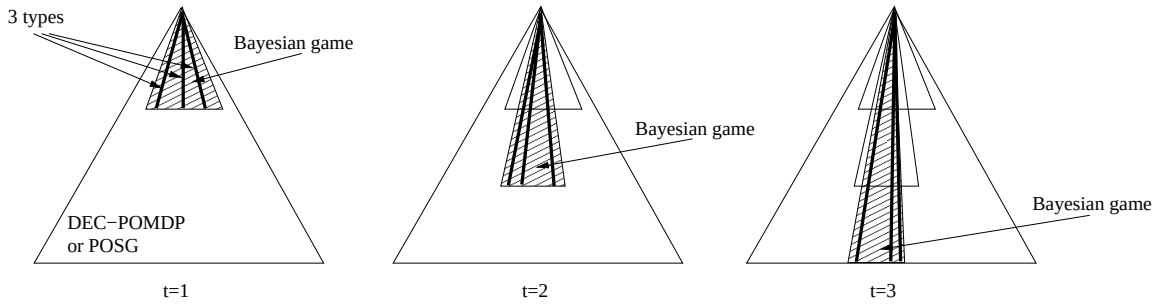

FIG. 5.3 – The iterative construction of three Bayesian games, approximating a given DEC-POMDP or POSG. Each game at iteration t samples a history of observations (=types) and actions until time t. Histories with small probabilities are discarded, thus keeping the set of types reasonably small.

The first step in solving this paradigma consists in establishing the Bayesian game at time t. This means essentially summarizing the local history of actions and observations into the type profiles. Given a prior distribution over the agents' types, a posterior distribution at time t can then be established. The second step consists in solving this game. Since there are still $(T-t)$ steps to go, an approximation of the expected future rewards is required, and the authors suggest to use one-step lookahead and the QMDP heuristic (see Section 3.2.5.1) to do so. The third step consists in simulating a single-step execution of the game, thus moving on one step further. Since the game is stochastic, the authors suggest that each agent simply samples a local observation and updates its own type accordingly. Based on the new type and the previously solved Bayesian game, each agent then choses an action, and the system transitions to a successor state. Since the number of observation histories increases exponentially with the horizon, histories - and thus types - with low probability are discarded, keeping the type space reasonably small. This is represented by narrowing the triangle that represents the Bayesian game's type space in Figure 5.3.

Solving games usually implies selecting one of several possible equilibria, and the Bayesian game approximation thus inherently needs a central controller to work. If however the question of equilibrium selection can be coordinated in some way, the authors claim that the algorithm can be distributed on the agents level. Given a commonly known prior over types, and given that the solution of each stage game is also known by all agents, each agent can update the type space, and a new probability distribution over new types, independently, and the results of this update can be guaranteed to be consistent. A sketch of the Bayesian games based solution technique,

proposed by [EMGST04], is given in Algorithm 27. As for all equilibrium based algorithms, the

Algorithm 27 `BayesianGamePolicyConstruction()`

Require: A DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$, a horizon T, and a start state s_0
Ensure: A joint equilibrium policy $\boldsymbol{\pi}$

```
for t=0 to T do
    Establish and solve Bayesian game at time t, leading to π^t = ⟨π_1^t, ..., π_n^t⟩
    for all agent i do
        Sample observation o_i^t, update type θ_i^t
        Select next action a_i^t = π_i^t(θ_i^t)
    end for
    Sample s^{t+1} from T(s^t, ⟨a_1, ..., a_n⟩, s^{t+1})
end for
```

solutions of the Bayesian game approach can only be guaranteed to be locally optimal. Also, since action selection and state transitioning is based on sampling, the resulting policies only constitute an approximative solution.

5.4 Conclusion

The underlying concepts of game theory, such as equilibrium solutions, best response strategies, or the alternation maximization principle, are essential to understanding the optimal decentralized control of dynamic systems. Although game theory considers a more general problem class, namely competitive scenarios, its solution concepts have been applied to cooperative multi-agent problems. The very central insight is that single agent policies cannot be evaluated independently one from each other, but that only the behavior of joint policies can adequately be qualified. A direct consequence is that multi-agent control problems are in general more difficult to solve than their single-agent counterparts, a property that is emphasized by the worst-case complexity of the DEC-POMDP model introduced in the previous section.

The alternation maximization principle and its variants consists in alternately maximizing the behavior of a single agent only, thus transforming a multi-agent problem into a sequence of single-agent problems. Its complexity is thus lower than the one of an optimal planning algorithm, but the resulting policies may obviously be suboptimal. It has nevertheless been shown to be useful, especially for solving larger problems. An overview of some of the basic algorithms for solving stochastic games, partially observable stochastic games, and identical payoff games are shown in Figure 5.4.

Because game theoretic algorithms cannot guarantee optimality in cooperative settings, we do not consider them in more detail in the following sections. We also do not include their performance in the experiments. Their concepts however are important for understanding decentralized control problems and will be used in the following planning algorithms.

	Subjective Coevolution, [CSC02]
Model	Subjective MDPs
Assumptions	Reactive policies, based on last observation. Decomposable reward function. Decomposable transition function.
Horizon	Infinite
Solution	Nash equilibrium
Collaboration	Cooperative (competitive)
Description	Application of the alternation maximization principle to subjective MDPs.

	JESP, DP-JESP, [NTY+03]
Model	MTDP, DEC-POMDP
Assumptions	-
Horizon	Finite
Solution	Nash equilibrium
Collaboration	Cooperative (competitive)
Description	Application of the alternation maximization principle to DEC-POMDPs.

	Bayesian Games Approximation, [EMGST04]
Model	DEC-POMDP, POSG
Assumptions	-
Horizon	Finite
Solution	Nash equilibrium
Collaboration	Cooperative (competitive)
Description	Application of the alternation maximization principle to DEC-POMDPs or POSGs. Includes the consideration of private information (types).

FIG. 5.4 – Solving DEC-POMDPs using game theory.

Chapitre 6

Solving DEC-POMDPs using Dynamic Programming

6.1 Introduction to Dynamic Programming

The term *dynamic programming* labels a class algorithms that gradually build a problem solution by solving smaller sub-problems instead of directly solving the entire problem as a whole [Bel57]. The general principle can easily be explained by the following example : assume you would have to ship a package from some location A to some other location C by passing through a third location B. There are three different roads that go from A to B, and four different roads that lead from B to C, which makes a total of $3 \times 4 = 12$ different ways to accomplish the task. A naive approach consists in comparing all these 12 candidates, and in choosing the best one out of them. A dynamic programming approach consists in recognizing that the problem can be divided, and that the shortest overall path is constituted of the shortest path between A and B, and the shortest path between B and C, which makes a total of 7 paths to consider.

Applying the principle of dynamic programming to Markov decision processes has led to numerous algorithms, and we already presented some of them in a previous chapter when we discussed techniques for solving MDPs and POMDPs (see Chapter 3). They were heavily based on the iterative evaluation of state value functions. We will now discuss how Bellman's principle of dynamic programming can be extended to decentralized control problems. We will see that the difficulty in establishing an appropriate multi-agent value function is the main obstacle to multi-agent dynamic programming. The exact and approximate algorithms we develop in the following sections are thus heavily policy-based.

6.2 Exact Dynamic Programming for DEC-POMDPs

Dynamic programming for MDPs can be divided into two different classes of algorithms, namely the family of value iteration and the family of policy iteration approaches. The same is true for POMDPs, since it has been shown that the value function, which is a priori defined over an infinite set of belief states, can always be represented by finite means. For DEC-POMDPs however, it is so far not known whether a value iteration algorithm can be formulated. This is partly due to the fact that one might have to consider several subjective value functions, one for each agent, since each agent usually has a different belief over the system. The nesting of reciprocal beliefs, mentioned in Chapter 4, is one of the reasons why value iteration algorithms

for decentralized control problems are particularly hard to establish.

6.2.1 Dynamic Programming based on Linear Programming

In the following section, we will present Hansen et al.'s dynamic programming algorithm for DEC-POMDPs, the first general dynamic programming algorithm for solving DEC-POMDPs optimally [HBZ04]. We begin by reviewing the basic parts of dynamic programming for single-agent POMDPs. The DP operator has two steps, namely the exhaustive generation of policies, and the pruning of dominated policies. The generation step takes a set of policy trees of horizon t as input, and builds the exhaustive set of policy trees of horizon $(t+1)$, such that each tree of the new set has only subtrees that appear in the original set. As explained earlier, this is a *bottom-up* policy construction. The pruning step then traverses the new policy set, examines whether each one of these policies is indeed useful, and eliminates dominated policy trees. This is done by analyzing the value function of the POMDP.

In the multi-agent case, we have to consider n policy sets and n value functions, but above all, we have to define them over the *multi-agent belief space*. Realizing how to adopt the notion of beliefs to multi-agent environments, without creating nested dependencies of reciprocal beliefs, was one of the major breakthroughs in extending dynamic programming to problems involving multiple decision makers. Once the notion of a multi-agent belief space has been accepted, the dynamic programming techniques can in principle be extended straightforwardly to n policy sets, and we will now detail what this means for the two steps of the dynamic programming operator.

Generating the set of policy trees for the next horizon is almost identical to what is done for POMDPs, except that it has to be done n times. Also the pruning step is essentially the same as in the mono-agent case, but it leads to different consequences. It consists in solving the linear program shown in Figure 6.1, the multi-agent extension of the one shown in Figure 3.8. If the re-

maximize	ϵ	
subject to	$V_i(\boldsymbol{b}_i, \tilde{q}_i) + \epsilon \leq V_i(\boldsymbol{b}_i, q_i)$	$(\forall \tilde{q}_i \neq q_i)$
with	$\sum_{s \in \mathcal{S}} \sum_{\boldsymbol{q}_{-i} \in Q_{-i}} \boldsymbol{b}_i(s, \boldsymbol{q}_{-i}) = 1$	
	$\boldsymbol{b}_i(s, \boldsymbol{q}_{-i}) \geq 0$	$(\forall s \in \mathcal{S})(\forall \boldsymbol{q}_{-i} \in Q_{-i})$

FIG. 6.1 – Linear program for identifying dominated multi-agent policies.

sult of the linear program is negative or equal to zero ($\epsilon \leq 0$), then the policy q_i is dominated and can be removed. The important difference is that removing a policy from the policy set of some agent i changes the belief space of all other agents. More precisely, if policy q_i is removed from the policy set Q_i of agent i, then agent j should not consider any belief where the probability of policy q_i is greater than zero. It is therefore possible that a policy q_j, which was useful in belief space B_j, is now completely dominated in the new belief space B'_j where the dimension of beliefs over policy b_i has been removed from. In a more general sense, the removal of a policy from one policy set can lead to the removal of a policy in another policy set and so on, which is why a single pruning step is in general not sufficient to compute a minimal representing of the optimal

value function. The fundamental difference in multi-agent dynamic programming hence is the need for *iterative pruning* until no more pruning is possible. The backup step and the iterative pruning step are visualized in Figure 6.2 for the case of a two agent problem with two actions and two observations. The dynamic programming approach for solving DEC-POMDPs is shown

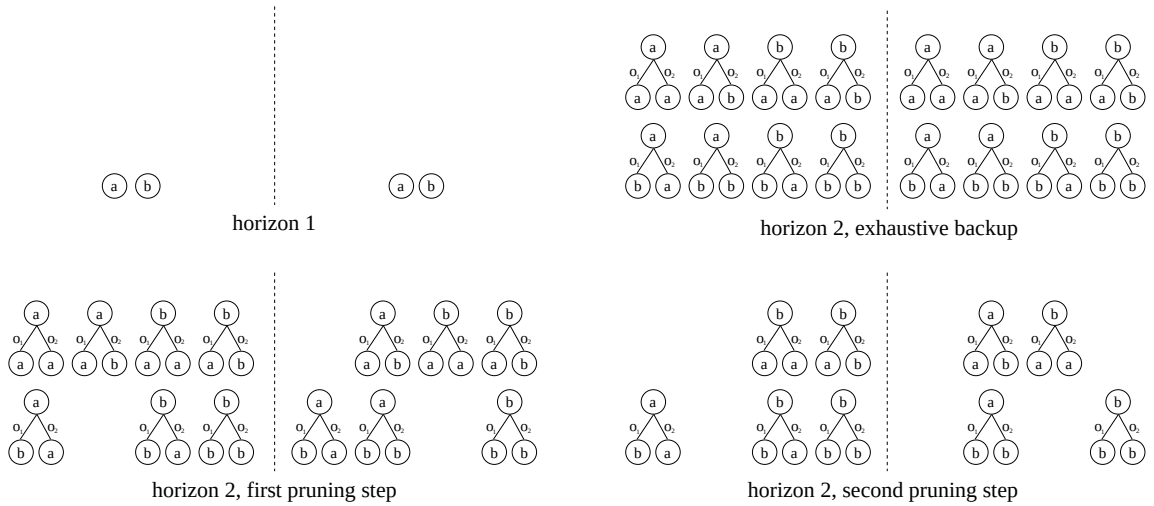

FIG. 6.2 – The exhaustive backup step and the iterative pruning step when passing from horizon 1 policies to horizon 2 policies. The second pruning step removes policies that were only needed as best-responses for those policies that were removed in the first pruning step. Additional pruning steps may be necessary in order to obtain a minimal policy set.

in Algorithm 28. It makes use of the same exhaustive policy generation procedure that was introduced in the context of POMDPs (Algorithm 18). For the special case of a *symmetric system*, where all agents are identical and thus entirely interchangeable, the question arises whether it is really necessary to keep n different policy sets, or whether they are not all identical. The answer is that the pruning step can lead to a "specialization" of the agents, and that it is thus necessary to keep the individual policy sets separately. In the RoboCupRescue domain for example, one could imagine two identical robots faced with a blocked road, where one agent could specialize in lifting the barrier, permitting the other agent to pass under it. The set of optimal policies for the second robot would thus depend on the policies identified for the first robot.

6.2.1.1 Solving POSGs

Hansen et al.'s multi-agent dynamic programming algorithm presents one particularity, namely that it is currently not exploiting the fact that agents are cooperative. The pruning step only removes policies that are not useful under any circumstances, which means they do not dominate the belief space in any region. This is true in cooperative as well as in competitive scenarios, which is why the same algorithm can also be applied to solve the competitive counterpart of the DEC-POMDP model, the partially observable stochastic game (POSG). The only difference lies in the final selection of an optimal joint policy. Whereas in a cooperative scenario, it is sufficient to maximize over all possible policy combinations to obtain the one that is optimal for the given start state distribution, choosing policies for the players of a game includes solving for an equilibrium point. The issue of identifying equilibria in games is intensively studied in game theory [Mye97, OR94].

Algorithm 28 Dynamic Programming for DEC-POMDPs - `DPforDEC-POMDPs()`

Require: A DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$ and a horizon T
Ensure: A set of useful policies for each agent

for all agent i **do**
  Initialize $Q_i^0 \leftarrow \emptyset$
end for
for $t = 1$ **to** $t = T$ **do**
  /* Exhaustive policy generation */
  for all agent i **do**
    $Q_i^t \leftarrow$ `ExhaustiveGeneration`(Q_i^{t-1})
  end for
  /* Iterative pruning */
  repeat
    Find an agent i and a policy $q_i \in Q_i^t$ with

$$(\forall \boldsymbol{b}_i)(\exists \tilde{q}_i \in Q_i^t \backslash \{q_i\}) \quad \text{such that} \quad V_i(\boldsymbol{b}_i, \tilde{q}_i) \geq V_i(\boldsymbol{b}_i, q_i)$$

    $Q_i^t \leftarrow Q_i^t \setminus \{q_i\}$
  until there is no more pruning possible
end for

6.2.1.2 Open Issues

We will now discuss some open problems and current research that is done to speed-up the dynamic programming approach presented above.

Parallelization One issue that arises naturally when dealing with distributed problems is that of parallelizing the computational effort. We believe indeed that the dynamic programming approach can be distributed, although this has not been attempted so far. In such a setting, each processor would store the policy set of one particular agent. The exhaustive policy generation step could be executed separately one each processor, but the resulting policy sets would have to be communicated in order to allow the construction of the appropriate belief spaces. The linear programming part itself then is inherently distributed, but in order to allow the iterative pruning step to work, the identified dominated policies would have to be communicated to all other processors. It is very likely that the dynamic programming algorithm can be efficiently implemented on a multi-processor machine with a shared memory.

Incremental pruning The major limitation of the dynamic programming approach is the explosion in memory requirements, since each step requires to first generate all policies for the next horizon, before beginning the step of pruning. This is similar to POMDPs, but appears to be far more limiting in DEC-POMDPs, because the individual value functions are now defined over much larger belief spaces that include all policies of all agents. In its current implementation, the dynamic programming algorithm thus quickly runs out of memory [HBZ04]. It would be an important step forward to apply the incremental pruning approaches known from single-agent POMDPs to the multi-agent case, and to show whether it is possible to solve larger problems.

Pruning rules The current pruning rule removes strategies that are *very weakly dominated*, which means strategies that do not dominate the belief space at all. However, other pruning rules

can be imagined, and it remains an open question whether it is possible to establish pruning rules that allow addressing issues such as

- pruning in cooperative settings - exploiting the fact that agents do not try to harm themselves
- approximated pruning - removing policies that are *almost* very weakly dominated in order to solve for larger problems, an issue that has already been addressed in the case of POMDPs [FH01]

In competitive scenarios, the concept of pruning policy sets translates to identifying the equilibrium points of a corresponding normal form game. Applying the theory of equilibrium identification to allow pruning for POSGs thus remains a challenge for improving the basic multi-agent dynamic programming algorithm.

Optimal ordering As we pointed out earlier, the removal of certain policies can lead to the removal of other policies that were only best response strategies to those policies already removed. The ordering in which dominated policies are identified hereby affects the amount of pruning possible, and the question arises whether there exists an optimal ordering - and whether it is efficiently computable - that produces a minimal policy set.

6.2.2 Point-based Dynamic Programming

We now introduce a variant of the above multi-agent dynamic programming approach that addresses two issues of Hansen et al.'s algorithm, namely

1. the computationally expensive linear programming part, necessary to identify dominated policy vectors, and
2. its current inability to exploit the cooperative nature of the problem setting.

This new *point-based multi-agent dynamic programming algorithm*, first presented in [SC06], is a synthesis of point-based approximations for single-agent POMDPs, and dynamic programming for multi-agent POMDPs. Instead of first generating all policies, and then pruning some of them, the point-based approach first identifies those belief points at which a policy is actually required. Large parts of the belief space will indeed never be visited because of the constraints of the transition function. Other parts can possibly be excluded because agents try to cooperate. What remains is a small fraction of the original multi-agent belief space, namely a finite number of candidate belief points. Determining the optimal policy at a given belief point is no longer a continuous problems ; it only requires maximizing over the set of candidate policies.

As for POMDPs, the point-based approach consists of two alternating phases. It first determines a discrete set of relevant belief points. The individual value functions are then evaluated at those belief points only, and a best-response policy for each point is determined. Since the best-response policies are by definition dominating policies, this process leads to a new policy set that contains only useful policies.

6.2.2.1 Determining Relevant Multi-agent Belief States

The determination of the relevant multi-agent belief points constitutes the basis of our dynamic programming algorithm. As opposed to the single-agent POMDP case, where a belief state can simply be obtained by stochastic sampling of actions, observations, and Bayes' theorem [PGT03], defining a multi-agent belief state also includes determining adequate distributions over the possible future policies of all remaining agents, the so called *mixed strategies*. In order to detail how

realistic beliefs over mixed strategies can be obtained, we explicitly separate beliefs over states and beliefs over strategies : we denote $\boldsymbol{b}_i^{\mathcal{S},t}$ agent i's belief over states $\mathcal{S}$ at time t, and $\boldsymbol{b}_i^{j,t}$ agent i's belief over agent j's strategies at time t. The complete belief state of agent i at time t then is $\boldsymbol{b}_i^t = \langle \boldsymbol{b}_i^{\mathcal{S},t}, \boldsymbol{b}_i^{1,t}, \ldots, \boldsymbol{b}_i^{i-1,t}, \boldsymbol{b}_i^{i+1,t}, \ldots, \boldsymbol{b}_i^{n,t} \rangle$.

In a single-agent POMDP, only a subset of all possible belief states can actually be encountered in real situations. This results from the restrictions imposed by the transition and the observation functions. In a cooperative multi-agent setting, the same holds for beliefs over strategies. We will now show why. Dynamic programming for Markov decision processes is a variant of backward induction, in the sense that the value function at iteration t represents the expected reward for the last t steps of the policy execution. Similarly, the set Q_i^t contains those sub-policies that will potentially be selected during the last t steps by agent i. Determining the usefulness of a policy q_i in Q_i^t thus translates to the question : what distributions over the strategies Q_{-i}^t of the other agents does agent i have to consider, given that $(T-t)$ time steps have already passed ? We emphasize on the point that agents are cooperative, which means they have full knowledge about the policies that were distributed prior execution. Together with the knowledge of the transition function, the observation function, the local history of observations, and Bayes' theorem, each agent can therefore compute a distribution over the possible local execution histories of each other agent. More formally, given $\mathcal{P}$, $\mathcal{O}$, the joint policy $\boldsymbol{q}^T$ distributed prior execution, and a local history of observations $h_i^{T-t} = (o_i^1, \ldots, o_i^{T-t})$, agent i can compute $P(h_j^{T-t} | h_i^{T-t}, \boldsymbol{q}^T)$, a probability distribution over the histories of observations probably encountered by each other agent j. This is illustrated in Figure 6.3, where the policies of two agents i and j are represented. In order to determine the usefulness of agent i's sub-policy q_i^t, one has to situate this sub-policy in a complete joint policy, determine the associated local history of observations for agent i, which is (a, o_1, a, o_1) in the case of the example, and compute the probabilities of all possible histories of agent j that are consistent with agent i's local observations.

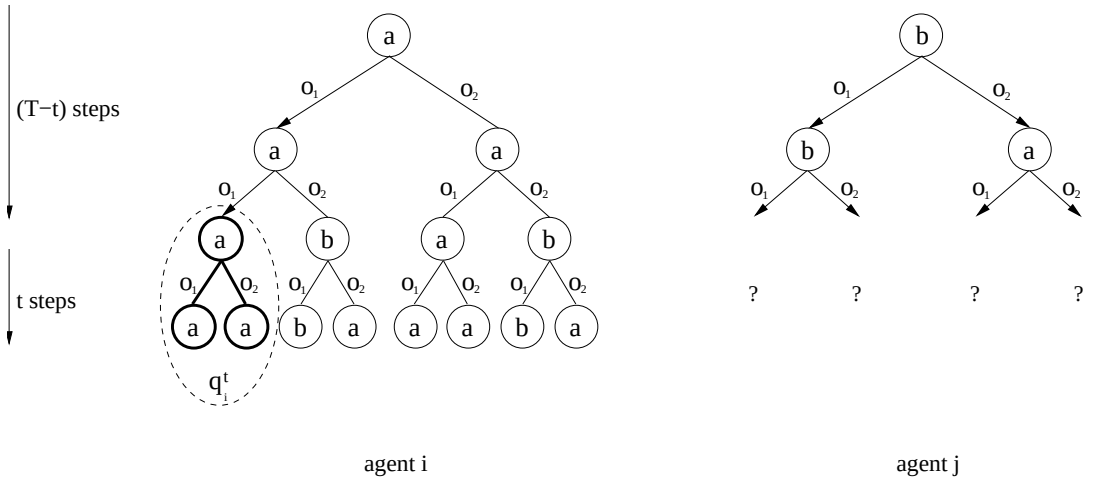

FIG. 6.3 – Determining the usefulness of agent i's sub-policy q_i^t includes reasoning about its place in the global policy structure, and Bayesian reasoning about possible prior histories of agent j until time $(T-t)$.

Since each observation history h_j^{T-t} of length $(T-t)$ corresponds to a path in policy tree q_j^T, it leads to one of its nodes at level $(T-t)$. The horizon-t sub-policy q_j^t attached at this node thus constitutes the policy that agent j is expected to execute under the assumption that it has observed the local history of observations h_j^{T-t} before. Since there is a one-to-one correspondence between local histories of observations and future policies, the belief of agent i about a particular strategy of agent j, given an observation history h_i^{T-t}, can be determined as

$$P(q_j^t|h_i^{T-t}, \boldsymbol{q}^{T-t}) = \begin{cases} P(h_j^{T-t}|h_i^{T-t}, \boldsymbol{q}^{T-t}), & \text{if } \boldsymbol{q}^T(h_j^{T-t}) = q_j^t \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

where $\boldsymbol{q}^T(h_j^{T-t})$ denotes the remaining sub-policy corresponding to observation history h_j^{T-t}. The complete belief of agent i over agent j's policies thus becomes

$$\boldsymbol{b}_i^{j,T-t}(q_j^t) = \sum_{h_i^{T-t}} P(q_j^t|h_i^{T-t}, \boldsymbol{q}^{T-t}) P(h_i^{T-t}|\boldsymbol{q}^{T-t}) \tag{6.2}$$

The estimation of the observation histories h_j^{T-t} can be carried out explicitly and at any time t during execution, because the system proceeds forward in time. The same however is not true during planning, since the planning process is oriented backwards. This means in particular that no *prior policy* $\boldsymbol{q}^{T-t}$ for the first $(T-t)$ time steps has been computed at iteration step t. Only the optimal policies for the last t steps have been determined. Evaluating the usefulness of a policy q_i^t thus involves four steps, namely

(a) determining a possible prior joint policy of actions $\boldsymbol{q}^{T-t}$,

(b) selecting a local history of observations h_i^{T-t} for agent i,

(c) determining a distribution over the possible histories of observations h_{-i}^{T-t} of all other agents, consistent with both $\boldsymbol{q}^{T-t}$ and $^{T-t}_{i}$, and

(d) computing a multi-agent belief state over policies and states.

We call *exact point-based multi-agent dynamic programming* the version of the algorithm that repeats the steps (a) to (d) for all possible configurations, and *approximate point-based multi-agent dynamic programming* the variant of the algorithm that only samples from these sets. The exhaustive belief set generation for the exact approach is summarized in Algorithm 29. It takes as input a prior joint policy for the first $(T-t)$ time steps, a local history of observations for agent i, and a *candidate set* of possible future policies for all other agents. The candidate set of policies is obtained from the set of useful policies of the previous iteration via exhaustive backup.

6.2.2.2 Determining the Best-response Policies

The second part of the point-based dynamic programming approach consists in determining the minimal set of dominating policies given the generated belief set. This step resembles the single-agent case : given an agent i, a set of possible belief points, and a set of candidate policies $\overline{Q}_i^t$, it means determining the best response policy for each belief as given in Definition 4.1.4.

The point-based multi-agent dynamic programming algorithm is summarized in Algorithm **??**. We also provide a theorem which states that the exact point-based multi-agent DP algorithm is indeed optimal.

Theorem 6.2.1. *The exact point-based multi-agent dynamic programming algorithm produces a complete set of useful policies for each agent.*

Algorithm 29 Exhaustive belief set generation - `ExhBeliefGen()`

Require: $\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{Q}_{-i}^t$
Ensure: The belief set $B(\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{Q}_{-i}^t)$
1: /* Compute the belief over states */
2: **for all** $s \in \mathcal{S}$ **do**
3: **for all** $\boldsymbol{h}_{-i}^{T-t}$ **do**
4: $\boldsymbol{b}_i^{\mathcal{S}}(s) \leftarrow \boldsymbol{b}_i^{\mathcal{S}}(s) + P(\boldsymbol{h}_{-i}^{T-t}|h_i^{T-t}, \boldsymbol{q}^{T-t})P(s|h_i^{T-t}, \boldsymbol{h}_{-i}^{T-t})$
5: **end for**
6: **end for**
7: /* Compute the beliefs over policies */
8: Let l be the number of leaves of $\boldsymbol{q}_{-i}^{T-t}$
9: **for all** selection of l policies from $\overline{Q}_{-i}^t$ **do**
10: **if** policy q_j is to be associated with the leaf corresponding to history h_j^{T-t} **then**
11: $\boldsymbol{b}_i^j(q_j) = P(\boldsymbol{h}_{-i}^{T-t}|h_i^{T-t}, \boldsymbol{q}^{T-t})$
12: **end if**
13: Add belief state $\boldsymbol{b}_i = (\boldsymbol{b}_i^{\mathcal{S}}, \boldsymbol{b}_i^1, \ldots, \boldsymbol{b}_i^n)$:

$$B(\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{Q}_{-i}^t) \leftarrow B(\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{Q}_{-i}^t) \bigcup \{\boldsymbol{b}_i\}$$

14: **end for**

Proof. We first emphasize that the belief set generated by Algorithm 29 is indeed exhaustive, which means that it generates all possible beliefs for a given local history of observations and the policy sets of all remaining agents. The point-based multi-agent dynamic programming operator (Algorithm 30) then generates the exhaustive sets of optimal policies for the given belief states. The proof is by induction : if each Q_i^{t-1} contains a complete set of useful policies for horizon $(t-1)$, then the belief set generation and the point-based multi-agent DP operator guarantee that each Q_i^t contains a complete set of useful policies for horizon t. The induction hypothesis is satisfied if $(\forall i)(Q_i^0 = \emptyset)$, and computations begins at $t = 1$. □

6.3 Approximate Dynamic Programming for DEC-POMDPs

The previous exact dynamic programming algorithms quickly either run out of memory or out of time, even when solving finite-horizon problems. In addition, it is known that solving infinite-horizon DEC-POMDP optimally is unfeasible in the most general case, since solving the single-agent POMDP over an infinite horizon is already undecidable [MHC03]. This is why approximate solutions have been identified, that enable the solution of larger finite-horizon, and infinite-horizon problems. We will present two of these approaches, which are based on the exact algorithms described before.

6.3.1 Bounded Policy Iteration

The major limitation of the general dynamic programming algorithm for DEC-PODMPs is the explosion in memory requirements : for each additional horizon, a superexponential number of new value functions has to be considered [HBZ04]. Instead of fixing the problem horizon and searching for an optimal policy, Bernstein et al. recently proposed an alternative approach based

Algorithm 30 Exact point-based multi-agent dynamic programming - `PBMADP()`

Require: A DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$ and a horizon T
Ensure: A set of useful policies for each agent

for all agent i **do**
  Initialize $Q_i^0 \leftarrow \emptyset$
end for
for $t = 1$ **to** $t = T$ **do**
  for all agent i **do**
    for all prior joint policy $\boldsymbol{q}^{T-t}$ **do**
      for all local history h_i^{T-t} **do**
        $B(\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{Q}_{-i}^t) = \texttt{ExhBeliefGen}(\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{Q}_{-i}^t)$
        for all $b_i^t \in B(\boldsymbol{q}^{T-t}, h_i^{T-t}, \overline{Q}_{-i}^t)$ **do**
          $Q_i^t \leftarrow Q_i^t \bigcup BR_i(b_i^t, \overline{Q}_i^t)$
        end for
      end for
    end for
    Set $\overline{Q}_i^t \leftarrow Q_i^t$
  end for
end for

on first bounding the size of the policy and then trying to optimize its parameters [BHZ05]. In this case, policies are based on finite state controllers, which can be executed for an infinite amount of time, and the algorithm thus constitutes one way of solving infinite-horizon DEC-POMDPs.

6.3.1.1 Correlated Stochastic Finite State Controllers

We have seen that optimal policies for finite-horizon POMDPs, whether they are centralized or distributed, can always be represented as deterministic decision trees. The common extension of decision trees to infinite-horizon problems are *finite state controllers*, and they have already been used for solving single-agent POMDPs [Han98]. A finite state controller is the extension of a finite state automaton and includes the selection of an action for each controller node. Specifying a finite state controller thus means defining a number of nodes, a node transition function, and an action selection function. The execution of the controller is started in some initial node, and the state of the controller then changes as a function of the incoming observations. At the same time, an action is selected.

Whereas an optimal solution for finite-horizon POMDPs can be represented as a deterministic policy, the same does not hold for the infinite-horizon case : *"In a POMDP, the best stationary stochastic policy can be arbitrarily better than the best stationary deterministic policy."* [SJJ94]. This means that the performance of a finite state controller can be improved by allowing stochasticity. An example for this is shown in figure 6.4.

In a multi-agent setting, an additional problem of synchronization arises when using stochastic policies. Consider for example the case where two agents can alter between the actions a and b, but where it is crucial that both agents chose the same action, leading or to aa or to bb [CB98, BHZ05]. Such a setting is shown in Figure 6.5. Without communication, nor any external coordination, it is not obvious how to achieve this synchronization. In such a case, an additional

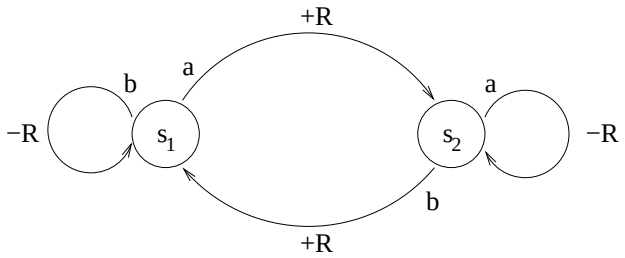

FIG. 6.4 – Shown is a POMDP with two states s_1 and s_2, two actions a and b, two possible rewards $+R$ and $-R$, but only one observation. The agent is thus unable to distinguish between states, and any deterministic policy would lead to infinitely accumulating a reward of $-R$. The optimal stationary stochastic policy would select action a and action b with probability 0.5 each, leading to an expected reward of 0.0 at each step. The example is taken from [SJJ94].

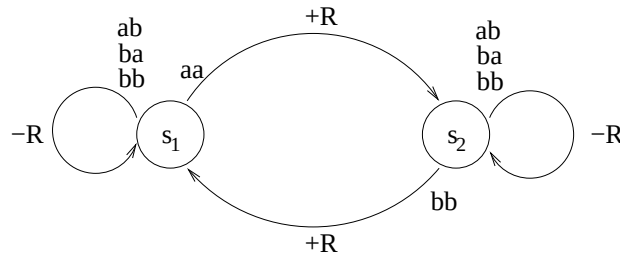

FIG. 6.5 – Shown is the multi-agent extension of the problem before. The optimal joint policy consists in selecting joint action *aa* and joint action *bb* with probability 0.5 each, while avoiding joint actions *ab* and *ba*. This synchronization can only be achieved via an external correlation device. The example is taken from [BHZ05].

additional source of information that both agents have access to can sometimes help. We refer to such a source as a *correlation device*, which evolves independently of the execution of the system. Goldman and Zilberstein call this *common uncontrollable features* [GZ04].

Definition 6.3.1 (Correlation Device). *We define a* correlation device *as a tuple* $\langle C, \psi \rangle$, *where*

- C *is a finite set of controller nodes,*
- $\psi : C \rightarrow \Delta C$ *is a stochastic node transition function.*

The need for correlation among multiple decision makers early appeared in the game theory literature, where the role of a central *mediator* has extensively been studied [Mye97]. The first ones to mention its usefulness for cooperative systems were maybe [PKMK00]. It is important to understand that a correlation device does not allow to pass any information between the agents. Also note that the special case of a single-node correlation device effectively means the absence of correlation.

Designing optimal stochastic policies for infinite-horizon multi-agent systems thus includes considering the presence of possible correlation devices. We formally define a *correlated stochastic finite state controller* as follows :

Definition 6.3.2 (Correlated Stochastic Finite State Controller). *A* correlated stochastic finite state controller *for agent i is a tuple $\langle N_i, \mathcal{A}_i, \Omega_i, \mathcal{C}, \phi_i, \eta_i \rangle$, where*

- *N_i is a finite set of controller nodes,*
- *$\mathcal{A}_i$ is a finite set of actions,*
- *Ω_i is a finite set of observations,*
- *$\mathcal{C}$ is a correlation device,*
- *$\psi_i : N_i \times \mathcal{C} \rightarrow \Delta\mathcal{A}_i$ is a stochastic action selection function,*
- *$\eta_i : N_i \times \mathcal{A}_i \times \Omega_i \times \mathcal{C} \rightarrow \Delta N_i$ is a stochastic node transition function.*

We will now describe a policy iteration algorithm that solves an infinite-horizon DEC-POMDP using policies based on correlated stochastic finite state controllers.

6.3.1.2 Bounded Policy Iteration

The policy iteration algorithm presented in [BHZ05] takes as input a fixed-size stochastic finite state controller per agent, plus an additional correlation device, and tries to improve the parameters ψ_i and η_i of each controller, as well as the transition function η of the correlation device. This process relies on evaluating a given correlated joint policy, and then improving one of its parameters by linear programming. Evaluating a joint policy can be done by solving the following system of linear equations

$$V(s, \boldsymbol{n}, c) = \sum_{\boldsymbol{a}} P(\boldsymbol{a}|\boldsymbol{n}, c) \left[R(s, \boldsymbol{a}) + \gamma \sum_{s', \boldsymbol{o}, \boldsymbol{n}', c'} P(s', \boldsymbol{o}|s, \boldsymbol{a}) P(\boldsymbol{n}', c'|\boldsymbol{n}, c, \boldsymbol{a}, \boldsymbol{o}) V(s', \boldsymbol{n}', c') \right] \quad (6.3)$$

where $P(\boldsymbol{a}|\boldsymbol{n}, c)$ represents the joint action selection function given by the local ψ_i's, $P(s', \boldsymbol{o}|s, \boldsymbol{a})$ stands for the system's transition and observation probabilities, and $P(\boldsymbol{n}', c'|\boldsymbol{n}, c, \boldsymbol{a}, \boldsymbol{o})$ represents the joint node transition function given by the local η_i's, and the transition function of the correlation device.

Improving the correlated joint policy means choosing one of the local controllers, and performing a *bounded backup* of its parameters ψ_i and η_i, such that the above value function of the system is globally improved. Performing a backup of a stochastic controller that globally improves its value function has first been described by Poupart and Boutilier for single-agent POMDPs [PB03]. The corresponding linear program for the multi-agent case is shown in Figure 6.6.

It can be proven that the bounded backup always leads to an improvement of the local controller [BHZ05]. However, since only one controller can be optimized at a time, the approach is only guaranteed to reach a local maximum of the global value function. This is obviously the common drawback of all optimization techniques that do not proceed by directly optimizing the joint behavior [CSC02, NTY+03].

6.3.2 Approximating Point-based DP

We now derive an approximate version of the exact point-based dynamic programming algorithm presented earlier. The key idea is to avoid the exhaustive generation of prior policies and belief states, required by Algorithm 30.

maximize ϵ

subject to $V(s, \boldsymbol{n}, c) + \epsilon \leq \sum_{\boldsymbol{a} \in \boldsymbol{A}} P(\boldsymbol{a}_{-i}|\boldsymbol{n}_{-i}, c) \Big[\alpha(a_i, c) R(s, \boldsymbol{a}) + \gamma \sum_{s', \boldsymbol{o}, \boldsymbol{n}', c'} P(s', \boldsymbol{o}|s, \boldsymbol{a}) \beta(n'_i, c, a_i, o_i) P(\boldsymbol{n}'_{-i}, c'|\boldsymbol{n}_{-i}, c, \boldsymbol{a}_{-i}, \boldsymbol{o}_{-i}) V(s', \boldsymbol{n}', c') \Big]$ $(\forall s)(\forall \boldsymbol{n}_{-i})(\forall c)$

with $\sum_{a_i \in \mathcal{A}_i} \alpha(a_i, c) = 1, \quad \sum_{n'_i \in N_i} \beta(n'_i, c, a_i, o_i) = \alpha(a_i, c)$ $(\forall c)(\forall n'_i)(\forall a_i)(\forall o_i)$

$\alpha(a_i, c) \geq 0, \quad \beta(n'_i, c, a_i, o_i) \geq 0$

FIG. 6.6 – Linear program for improving the parameters of node n_i of agent i's finite state controller.

6.3.2.1 Generation of Joint Policies

Generating the exhaustive set of prior policies is often impractical, because the set of possible policies grows more than exponentially with the horizon and the number of agents. We adopt a strategy that has already been used in the single-agent case in order to determine candidate belief states [PGT03] : we sample from the set of possible prior policies. In the multi-agent case, this translates to sampling from the set of entire joint policies. Sampling can be done in different ways, and one problem is to identify those policies that are the most representative for the entire candidate set. Pineau et al. have focused on samples that are spread out as far as possible in the belief space. The problem is that such an approach would require to first generate all prior policies. One way to avoid this computation is to look for policies that are spread out in policy space and not in belief space, for example by using the Manhattan distance as a simple metric for policy trees : each two nodes in two different trees but at the same position have distance 0 if the contain the same action, and have distance 1 otherwise. It might however occur that two policies, that are far away one from each other in policy space, might be very close in the corresponding belief space.

6.3.2.2 Generation of Belief States

The complexity of generating all possible belief states is due to the exponentially many possible assignments of the policies in $\overline{Q}^t_{-i}$ to the leaves of $\boldsymbol{q}^{T-t}$. Some leaves however are much less likely to be visited than others, and we can specify the error we possibly commit if we avoid testing all possible policy subtrees at that leaf node, but rather assign a subtree at random. For a given history h_i of agent i, we denote γ the probability of some compatible vector of histories $\boldsymbol{h}_{-i}$ for all agents but i :

$$\gamma = P(\boldsymbol{h}_{-i}|h_i, \boldsymbol{q}^{T-t}) \tag{6.4}$$

The maximal loss of rewards we can expect when the worst possible policies are assigned to the leaves corresponding to $\boldsymbol{h}_{-i}$ is $\gamma(T-t)(R_{max} - R_{min})$. If we allow an error of ϵ, then we can

avoid considering all possible assignments of policies to the leaves $\boldsymbol{h}_{-i}$ if

$$\gamma \leq \frac{\epsilon}{(T-t)(R_{max} - R_{min})} \tag{6.5}$$

6.3.2.3 Approximate Multi-agent Point-based Dynamic Programming

The approximate multi-agent point-based dynamic programming algorithm consists in a combination of sampling prior joint policies and sampling relevant belief states. It remains an open question whether there exist a general error bound for this approach.

One might expect that the approximate approach will produce an uncomplete set of dominating policies, eventually leaving out some optimal policies that would have been identified by the optimal algorithm. This however is not true. The approximate approach might actually return dominated policies, simply because a missing policy in the policy set of some agent leads to a modified belief space where a policy might appear to be dominating, although in the optimal belief space this is not the case. We confirm this fact by the following theorem.

Theorem 6.3.1. *The policy sets produced by the approximate point-based multi-agent dynamic programming algorithm may be incomplete, and they may contain dominated policies that do not appear in the resulting sets of the optimal algorithm.*

Proof. We first argue that the resulting policy sets may be incomplete. This is essentially due to the way belief states are sampled. We then show that the resulting policies may be dominated. This is due to the incompleteness of the policy sets : given an incomplete set of policies Q_i^{t-1}, the set of candidate policies $\overline{Q}_i^t$ for the next horizon is necessarily incomplete as well. This might result in a policy $q_i^t \in \overline{Q}_i^t$ that appears to be optimal at some given belief state, but that is actually dominated by another policy $\tilde{q}_i^t \notin \overline{Q}_i^t$ which could not be considered. □

6.4 Conclusion

The dynamic programming algorithm developed by Hansen et al. [HBZ04] was the first optimal non-trivial planning algorithm for finite-horizon DEC-POMDPs. It also provided the necessary insights that led to the new point-based dynamic programming approaches, a synthesis of single-agent point-based algorithms and the multi-agent dynamic programming approach. The bounded policy iteration is a straightforward synthesis of bounded policy iteration for POMDPs, and decentralized decision making, introducing the novel concept of a correlation device. A summary of the dynamic programming algorithms that we introduced in this chapter can be found in Figure 6.7.

The main bottleneck of the continuous dynamic programming algorithm is memory space. In its current version, the algorithm first has to generate all candidate policies for the next horizon before starting pruning. More sophisticated pruning techniques, such as the incremental pruning approaches developed for POMDPs, might help reducing this memory bottleneck. The main bottleneck of the point-based algorithms is time, namely the time necessary to enumerate all potential belief states. However, not each belief state witnesses a different optimal policy, and a smarter enumeration technique might for example help reducing the number of belief states necessary in order to compute an optimal policy.

	Exact DP, [HBZ04]
Model	DEC-POMDP, POSG
Assumptions	-
Horizon	Finite
Solution	Optimal (dominating) policies
Collaboration	Cooperative (competitive)
Description	Dynamic programming on policy trees.

	Exact Point-based DP, [SC06]
Model	DEC-POMDP
Assumptions	-
Horizon	Finite
Solution	Optimal (dominating) policies
Collaboration	Cooperative
Description	Point-based dynamic programming on policy trees using multi-agent belief states.

	Bounded Policy Iteration, [BHZ05]
Model	DEC-POMDP, POSG
Assumptions	-
Horizon	Infinite
Solution	Approximation (based on controller size)
Collaboration	Cooperative (competitive)
Description	Dynamic programming on joint policies represented as stochastic finite automata.

	Approximate Point-based DP, [SC06]
Model	DEC-POMDP
Assumptions	-
Horizon	Finite
Solution	Approximation (based on belief state and policy sampling)
Collaboration	Cooperative
Description	Point-based dynamic programming on policy trees using sampling of multi-agent belief states.

FIG. 6.7 – Solving DEC-POMDPs using dynamic programming.

Chapitre 7

Solving DEC-POMDPs using Heuristic Search

7.1 Introduction to Heuristic Search

Searching for optimal policies is the complimentary approach to policy computation via dynamic programming. While current dynamic programming algorithms are bottom-up approaches, we have seen in Chapter 3 that an equivalent top-down heuristic search method can be established for POMDPs. The work that we present in this chapter extends heuristic methods to DEC-POMDPs. It has partly been motivated by the following insight of Hansen et al. in [HBZ04] : *"Whether some kind of forward search can be done in the multi-agent case is an important open problem"*.

Optimal search methods have been studied for a long time in graph theory, and have since been extended to a variety of problem classes where an appropriate search space can be defined. We focus our attention on heuristic search, a variant of best-first search with an additional heuristic function that guides the selection of the node from which to start the exploration. In the following sections, we will first present classical heuristic A* search, mention the issues that have to be dealt with when extending this search technique to multi-agent domains, and finally present a finite-horizon and an infinite-horizon version of our multi-agent A* search algorithm MAA*.

7.2 Multi-agent Heuristic Search

The original multi-agent A* algorithm for solving finite-horizon DEC-POMDPs has first been presented in [SCZ05]. It is the first general and optimal attempt to apply heuristic search techniques for solving multi-agent decision problems. We will now motivate the use of A* search in general, and then describe MAA* in detail.

7.2.1 A* Search

Using the family of A* search algorithms to the problem of optimal planning seems to be very appealing for the following reason : it has been shown that, under certain conditions, A* evaluates the minimal number of nodes among all optimal search algorithms for which the same heuristic is used [DP85]. This is what is called *optimal efficiency*. This property obviously makes A* a

very appealing technique, and numerous research efforts have since been undertaken to extend the basic algorithm further : AO* for example searches in the space of trees rather than in the space of paths [Nil80], and we have shown its application to solving POMDPs in Section 3.2.3. LAO* finally is able to find solutions with loops [HZ01].

We will not go into the details of classical A* search itself, since we already presented its application to POMDPs in Section 3.2.3. For further information, we refer the reader to the textbook by Judea Pearl [Pea90]. Our main concern is how the principle of A* and AO* can be extended to dealing with multiple decision makers. The two issues we have to address are :

– identifying an appropriate search space
– defining admissible heuristics.

While the second issue, namely identifying heuristic functions that estimate the execution of a DEC-POMDP, can easily be established using simpler underlying processes (see Section 7.2.3), the first issue is by far more challenging. A naive attempt would be to extend the AO* search tree shown in Figure 3.10 to joint actions. Such an attempt is shown in Figure 7.1. The tree obviously

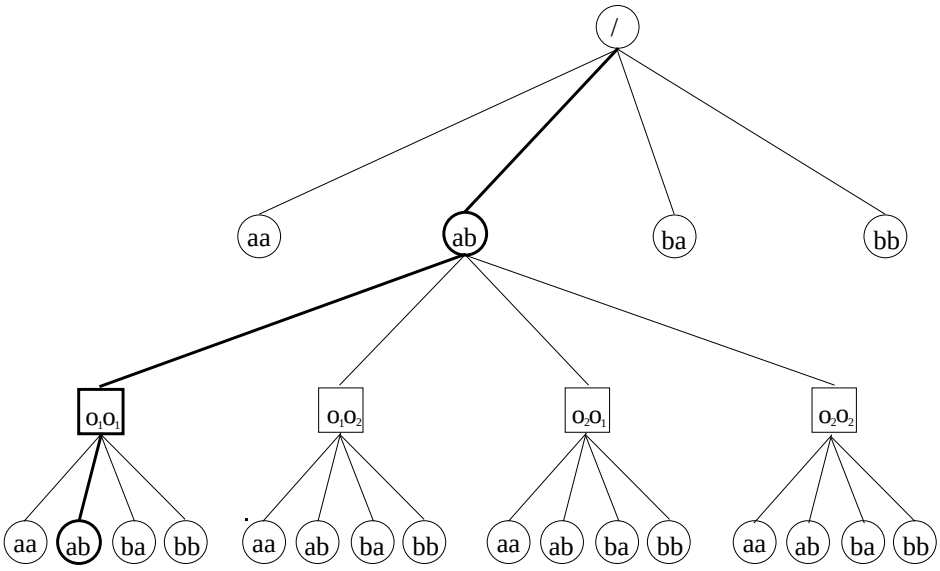

FIG. 7.1 – A section of an invalid multi-agent search tree.

searches through all possible joint actions and joint observations. It is however impossible to evaluate its nodes. Let us consider one of the leaf nodes of the tree, joint action $\langle a, b\rangle$, which follows joint observation $\langle o_1, o_1\rangle$, and joint action $\langle a, b\rangle$, indicated in bold. Since actions and observations are not shared between agents, agent 1 has the following perspective of the situation : it has perceived observation o_1, and has executed action a thereafter. This means that all joint actions that execute a after o_1 also have to be considered, since they are indistinguishable to agent 1, namely action $\langle a, a\rangle$ from the branch of subtree $\langle o_1, o_1\rangle$, and actions $\langle a, a\rangle$ and $\langle a, b\rangle$ from the branch of subtree $\langle o_1, o_2\rangle$. At the same time, this also means that all joint actions that execute b after o_1 have to be discarded, namely actions $\langle b, a\rangle$ and $\langle b, b\rangle$ from the branch of subtree $\langle o_1, o_1\rangle$, and actions $\langle b, a\rangle$ and $\langle b, b\rangle$ from the branch of subtree $\langle o_1, o_2\rangle$. Consider now the perspective of agent 2 : is has perceived observation o_1, and has executed action b thereafter. This means that all joint actions that execute b after o_1 also have to be considered, namely action $\langle b, b\rangle$ from the branch of subtree $\langle o_1, o_1\rangle$, and actions $\langle a, b\rangle$ and $\langle b, b\rangle$ from the branch of subtree

$\langle o_2, o_1 \rangle$. At the same time, this also means that all joint actions that execute a after o_1 have to be discarded, namely actions $\langle a, a \rangle$ and $\langle b, a \rangle$ from the branch of subtree $\langle o_1, o_1 \rangle$, and actions $\langle a, a \rangle$ and $\langle b, a \rangle$ from the branch of subtree $\langle o_2, o_1 \rangle$. The two situations are shown in Figure 7.2. It is not clear how these nested dependencies can be resolved in order to establish a correct

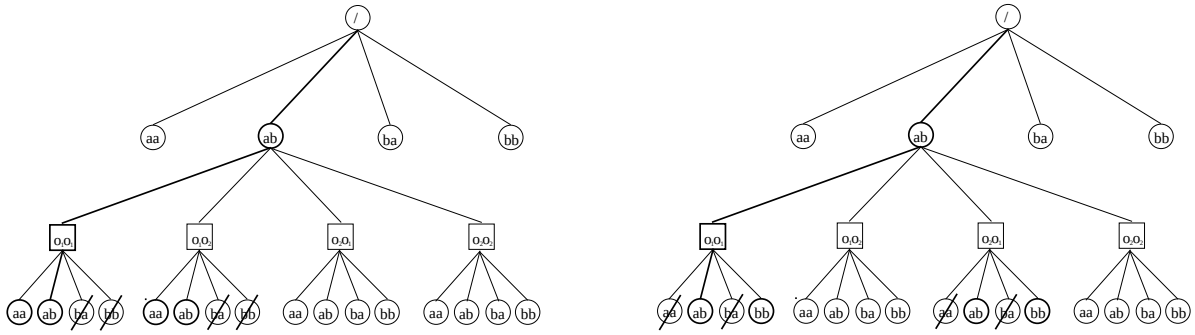

FIG. 7.2 – The two different subjective views of agent 1 and agent 2 respectively, when considering the evaluation of joint action $\langle a, b \rangle$ that followed joint action $\langle a, b \rangle$ and joint observation $\langle o_1, o_1 \rangle$ in the tree of Figure 7.1 : each agent makes different assumptions about the correlation of this node with the joint actions of other nodes of the search tree.

evaluation of leaf node $\langle a, b \rangle$. At the same time, they lead to contradictory requirements, where the inclusion or exclusion of certain branches depends on the subjective view of the situation. A different approach is needed. It is based on the evaluation of entire joint policies.

7.2.2 Extending A* Search to Multi-agent Domains

We have seen that a solution to a finite horizon POMDP can be represented as a decision tree, where nodes are labeled with actions and arcs are labeled with observations. We have also seen that a similar solution to an n-agent horizon-T DEC-POMDP with known start state can be formulated as a joint policy of n horizon-T trees, one for each agent, which are then executed synchronously. Forward search in the space of joint policies can be seen as an incremental construction of a set of horizon-$(t+1)$ joint policies from a parent horizon-t joint policy, which means expanding the leaf nodes of each policy tree in the horizon-t joint policy.

We build our approach on the popular A* algorithm as a basis for heuristic best-first search. Similarly to A* search, we are able to progressively build a search tree in the space of joint policies, where nodes at some depth t of the tree constitute partial solutions to our problem, namely joint policies of horizon t. Each iteration of the search process includes evaluating the leaf nodes of the search tree, selecting the node with the highest evaluation, and expanding this node and thus descending one step further in the tree. A section of such a multi-agent search tree is shown in Figure 7.3.

Heuristic search is based on the decomposition of the evaluation function into an exact evaluation of a partial solution, and a heuristic estimate of the remaining part. We introduce Δ^{T-t} as a *completion* of an arbitrary depth-t joint policy, which means a set of depth-$(T-t)$ joint policies that can be attached at the leaf nodes of a joint policy $\boldsymbol{q}^t$ such that $\langle \boldsymbol{q}^t, \Delta^{T-t} \rangle$ constitutes a complete joint policy of depth T. This allows us to decompose the value of any depth-T joint

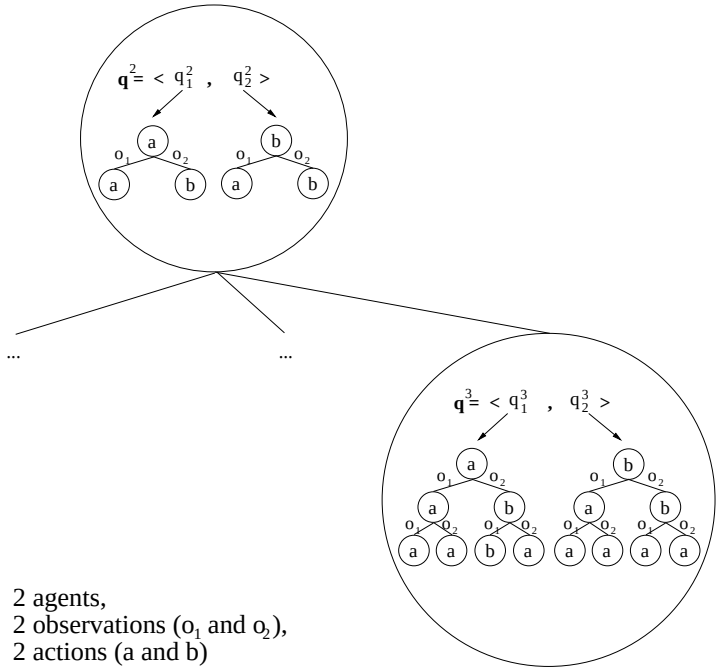

FIG. 7.3 – A section of the multi-agent A* search tree, showing a horizon 2 joint policy with one of its expanded horizon 3 child nodes.

policy into the value of its depth-t root policy ($t \leq T$), and the value of the completion :

$$V(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle) = V(s_0, \boldsymbol{q}^t) + V(\Delta^{T-t} | s_0, \boldsymbol{q}^t) \tag{7.1}$$

The value of a completion obviously depends on the previous execution of the root policy and the underlying state distribution at time t. Instead of calculating its value effectively, we are interested in estimating it efficiently. Like in A*, we set

$$F^T(s_0, \boldsymbol{q}^t) = V(s_0, \boldsymbol{q}^t) + H^{T-t}(s_0, \boldsymbol{q}^t) \tag{7.2}$$

as the *value estimate* for the joint policy of depth-t policy trees $\boldsymbol{q}^t$ and start state s_0. The function H must be an *admissible heuristic*, which means it must be an overestimation of the actual expected reward for any completion of joint policy $\boldsymbol{q}^t$

$$\forall \Delta^{T-t} : \quad H^{T-t}(s_0, \boldsymbol{q}^t) \geq V(\Delta^{T-t} | s_0, \boldsymbol{q}^t) \tag{7.3}$$

We will now develop multi-agent A* as an extension of classical A* search with the nodes of the search tree being constituted of partially developed joint policies.

7.2.3 The Heuristic Function

The core part of the search algorithm remains the definition of an admissible heuristic function. As pointed out by [LCK95] and later by [Hau00] for the single-agent case, an upper bound for the value function of a POMDP can be obtained through the underlying completely observable MDP. We have seen that the value function of a POMDP is defined over the belief space of the

system. The optimal value for a belief state $\boldsymbol{b}$ can then be approximated as follows :

$$V^*_{POMDP}(\boldsymbol{b}) \leq \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V^*_{MDP}(s) \tag{7.4}$$

We will now extend this property to decentralized systems, and show that there exists a similar and easily computable bound for DEC-POMDPs. We will in fact introduce a whole class of admissible heuristics and prove that they all overestimate the actual expected reward. In order to do so, we need some more definitions. We set

- $P(s|s_0, \boldsymbol{q})$ as the probability of being in state s after executing the joint policy $\boldsymbol{q}$ from s_0
- $h^t(s)$ as an optimistic *value function heuristic* for the expected sum of rewards when executing the best joint policy of depth t from state s

$$h^t(s) \geq V^{*t}(s) \tag{7.5}$$

  with $h^0(s) = 0$

This allows us to define the following class of heuristic functions :

$$H^{T-t}(s_0, \boldsymbol{q}^t) = \sum_{s \in \mathcal{S}} P(s|s_0, \boldsymbol{q}^t) h^{T-t}(s) \tag{7.6}$$

Intuitively, any such heuristic is optimistic because it effectively simulates the situation where the real underlying state is revealed to the agents after execution of joint policy $\boldsymbol{q}^t$.

Theorem 7.2.1. *Any heuristic function H as defined in (7.6) is admissible, if h is admissible.*

Proof. In order the prove the claim, we need to clarify what happens after execution of joint policy $\boldsymbol{q}^t$: each agent i will have executed its policy tree q_i down to some leaf node as a result of a sequence $\theta_i = (o_i^1, \ldots, o_i^t)$ of t observations. The completion $\Delta^{T-t}(\theta_i)$ then contains the remaining depth-$(T-t)$ policy tree agent i should execute afterwards. Similarly, $\theta = (\theta_1, \ldots, \theta_n)$ represents the joint observation histories for all agents, and the set of depth-$(T-t)$ policy completions for the whole team can thus be written as $\Delta^{T-t}(\theta)$. Its value depends on the underlying state distribution corresponding to the set of observation sequences θ. We can write for the value of any policy completion :

$$\begin{aligned}
V(\Delta^{T-t}|s_0, \boldsymbol{q}^t) &= \sum_{\theta \in \Theta^t} P(\theta|s_0, \boldsymbol{q}^t) V(\Delta^{T-t}(\theta)|s_0, \boldsymbol{q}^t) \\
&= \sum_{\theta \in \Theta^t} P(\theta|s_0, \boldsymbol{q}^t) \left[\sum_{s \in \mathcal{S}} P(s|\theta) V(s, \Delta^{T-t}(\theta)) \right] \\
&\leq \sum_{\theta \in \Theta^t} P(\theta|s_0, \boldsymbol{q}^t) \left[\sum_{s \in \mathcal{S}} P(s|\theta) V^{*T-t}(s) \right] \\
&= \sum_{s \in \mathcal{S}} \sum_{\theta \in \Theta^t} P(s|\theta) P(\theta|s_0, \boldsymbol{q}^t) V^{*T-t}(s) \\
&= \sum_{s \in \mathcal{S}} P(s|s_0, \boldsymbol{q}^t) V^{*T-t}(s) \\
&\leq \sum_{s \in \mathcal{S}} P(s|s_0, \boldsymbol{q}^t) h^{T-t}(s) = H^{T-t}(s_0, \boldsymbol{q}^t)
\end{aligned}$$

□

The computation of a good heuristic function is in computationally less expensive than the calculation of the exact value. In our case, the benefit lies in the reduction of the number of evaluations from $|\Omega^{tn}|$, which is the number of possible observation sequences of length t, to $|\mathcal{S}|$, the number of states : the value of the heuristic function $h^{T-t}(s)$ has to be calculated only once and should be recorded, since it can be reused for all nodes at level $(T-t)$ as stated in Equation (7.6). The computation of the value function heuristic h may lead to further savings, and we present several ways to obtain an admissible value function heuristic for our problem.

7.2.3.1 The MDP Heuristic

An easy way to calculate a value function heuristic is to use the solution of the underlying centralized MDP with remaining finite horizon $T-t$:

$$h^{T-t}(s) = V^{T-t}_{MDP}(s) \tag{7.7}$$

Solving an MDP can be done using any DP or search technique and requires only polynomial time. Using the underlying MDP as an admissible heuristic for search in POMDPs has already been applied to the single-agent case as described in [Was96] and later in [GB98]. In our case, the underlying MDP is the centralized and fully observable version of the initial problem, which means it is based on the real system states and joint actions.

7.2.3.2 The POMDP Heuristic

A tighter yet more complex value function heuristic constitutes in using the solution to the underlying partially observable MDP :

$$h^{T-t}(s) = V^{T-t}_{POMDP}(s) \tag{7.8}$$

Although the underlying POMDP is partially observable, it still considers joint actions and thus overestimates the expected sum of rewards for the decentralized system. We have seen that solving POMDPs is PSPACE-complete and usually involves linear programming. However, any upper bound approximation to the exact POMDP solution such as the *fast informed bound* method [Hau00] or any search method can also be used as an admissible heuristic.

7.2.3.3 Recursive Heuristics

The closer the heuristic is to the optimal value function, the more pruning is possible in the search tree. An important special case of a value function heuristic thus constitutes the optimal value itself : $h^t(s) = V^{*t}(s)$. It is the tightest possible heuristic :

$$h^{T-t}(s) = V^{*T-t}(s) \tag{7.9}$$

One way to compute this value efficiently is to apply heuristic search again. This leads to a recursive approach, where a search for a horizon T problem invokes several searches for horizon $(T-1)$ problems, and where each of them launches new subsearches. At each leaf node of the search tree, recursive search thus starts $|\mathcal{S}|$ new search subproblems of horizon $(T-t)$.

In general, there exists a tradeoff between the increased complexity of a tighter heuristic function, and the possible pruning it allows in the search tree. Determining whether it is beneficial or not to use a tighter heuristic function can be very difficult, since it usually requires a complete analysis of the influence of the problem parameters on the search process. We will not address this issue in our work.

7.2.4 The MAA* Algorithm

We now define the *multi-agent heuristic search algorithm* MAA*. The root of the search tree is initialized with the complete set of horizon-1 joint policies, the set $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, and the search then proceeds similarly as A* by expanding the leaf nodes in best-first order until an optimal horizon-T solution has been identified. Expanding a joint policy $\boldsymbol{q} = \langle q_1, \ldots, q_n \rangle$ means : for each leaf node in q_i, construct $|\Omega_i|$ child nodes and assign them an action. For a given tree of depth t, there are Ω^t new child nodes and thus $\mathcal{A}^{\Omega^t}$ different possible assignments of actions. For a joint policy of size n, there will thus be a total number of $(\mathcal{A}^{\Omega^t})^n$ new child nodes.

7.2.4.1 Resolving ties

In classical A* search, nodes are always fully expanded : for a given leaf node, all child nodes are immediately added to the so called *open list* D. In our case however, this approach presents a major drawback. As pointed out by [HZ96], suboptimal solutions that are found during the search process can be used to prune the search tree. Since our problem has more than exponential complexity, evaluating the search tree until depth $(T-1)$ can be considered as being "easy" : almost all the computational effort is concentrated in the last level of the tree. This means that suboptimal solutions will be found very early in the search process, and it is thus beneficial not to wait until all leaf nodes of depth T have been evaluated : if a suboptimal solution node with the same value as its parent is found, enumeration of the remaining child nodes can be stopped. Nodes are thus expanded incrementally, which means that only one child assignment of actions is constructed in each iteration step. The parent node remains in the open list as long as it is not fully expanded. Only if the same parent joint policy is selected in a further step, the next possible assignment of actions is evaluated. Together with the tie-breaking property, this may lead to savings in both memory and runtime.

7.2.4.2 Anytime search

Since solving a DEC-POMDP optimally may require a significant amount of time, it is sometimes preferable to obtain a near optimal solution as quickly as possible, which can then gradually be improved. As already mentioned above, suboptimal solutions are discovered early in the search. It is therefore straightforward to give the algorithm a more *anytime* character by simply reporting any new suboptimal solution that has been evaluated before it is added to the open list.

The discovery of suboptimal solutions in heuristic search can further be sped up through *optimistic weighting* : by introducing a weight parameter $w_i < 1$ on the heuristic, an idea first described by [Poh73], we are able to favor those nodes of the search tree that appear to lead to a suboptimal solution very soon, giving the search a more depth-first character :

$$\overline{F}_i^T(s_0, \boldsymbol{q}^t) = V(s_0, \boldsymbol{q}^t) + w_i H^{T-t}(s_0, \boldsymbol{q}^t) \tag{7.10}$$

Although the convergence criterion remains the F-value, expansion of the tree is now guided by the weighted $\overline{F}$-value, which may in addition be time dependent.

7.2.4.3 MAA*

The generalized MAA* algorithm with incremental node expansion and tie-breaking is summarized in Algorithm 31. We can prove that MAA* indeed returns an optimal solution for any finite horizon DEC-POMDP :

Algorithm 31 Multi-agent A* - `MAA*()`

Require: A DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$, a start state s_0, and a horizon T
Ensure: An optimal joint policy
1: Initialize open list $D = \times_i \mathcal{A}_i$
2: $\boldsymbol{q}_{temp} \leftarrow \arg\max_{\boldsymbol{q} \in D} F^T(s_0, \boldsymbol{q})$
3: **repeat**
4: /* Expand current best solution */
5: Select $\boldsymbol{q}^* \leftarrow \arg\max_{\boldsymbol{q} \in D} F^T(s_0, \boldsymbol{q})$
6: $\boldsymbol{q}^{*'} \leftarrow$ `Expand(`$\boldsymbol{q}^*$`)`
7: /* Check for new suboptimal solution */
8: **if** $F^T(s_0, \boldsymbol{q}_{temp}) < F^T(s_0, \boldsymbol{q}^{*'})$ **then**
9: $\boldsymbol{q}_{temp} \leftarrow \boldsymbol{q}^{*'}$
10: Output $\boldsymbol{q}^{*'}$
11: **for all** $\boldsymbol{q} \in D$ **do**
12: **if** $F^T(s_0, \boldsymbol{q}) \leq F^T(s_0, \boldsymbol{q}^{*'})$ **then**
13: $D \leftarrow D \setminus \{\boldsymbol{q}\}$
14: **end if**
15: **end for**
16: **end if**
17: /* Add new joint policy to open list */
18: $D \leftarrow D \bigcup \{\boldsymbol{q}^{*'}\}$
19: **if** $\boldsymbol{q}^*$ is fully expanded **then**
20: $D \leftarrow D \setminus \{\boldsymbol{q}^*\}$
21: **end if**
22: **until** $\exists \boldsymbol{q}^T \in D$ such that $\forall \boldsymbol{q} \in D : F^T(s_0, \boldsymbol{q}) \leq F^T(s_0, \boldsymbol{q}^T) = V(s_0, \boldsymbol{q}^T)$

Theorem 7.2.2. *MAA* is both complete and optimal.*

Proof. MAA* will eventually terminate in the worst case after enumerating all possible horizon-T joint policies, which means after constructing the complete search tree of depth T. The leaf node with the highest F-value then contains an optimal solution to the problem. If MAA* terminates and returns a joint policy $\boldsymbol{q}^T$, the convergence property of A* and the admissibility of the heuristic H guarantees the optimality of the solution. □

Anytime MAA* can be obtained by replacing the admissible evaluation function F in step 5 of the algorithm with a non-admissible evaluation function $\overline{F}$ as given in (7.10). This is shown in Algorithm 33.

MAA* searches in policy space : evaluation and exploration is based on joint policies. We have seen that other evaluation methods within the theory of Markov decision processes are based on state values, such that an optimal policy can be extracted by selecting actions that maximize the value of the current state. However, it is not clear if state values can be defined in a similar way for distributed systems with different partial information. Whether some sort of search in state space or belief space is possible for decentralized POMDPs thus remains an important open problem.

Algorithm 32 `Expand()`

Require: A joint policy $\boldsymbol{q} = \langle q_1, \ldots, q_n \rangle$ of depth t
Ensure: A joint policy $\boldsymbol{q}' = \langle q_1, \ldots, q_n \rangle$ of depth $t+1$
1: Attach $n|\Omega|^t$ new child nodes to $\boldsymbol{q}$
2: Associate each new node with an action. This should be done in a way as to avoid duplicates : each time `Expand()` is called on some joint policy, a different distribution of actions should be chosen.

Algorithm 33 Anytime multi-agent A* - `AnytimeMAA*()`

5 : Select $\boldsymbol{q}^* \in D$ such that $\forall \boldsymbol{q} \in D : \overline{F}^T(s_0, \boldsymbol{q}) \leq \overline{F}^T(s_0, \boldsymbol{q}^*)$

7.2.4.4 Solving Uniform DEC-POMDPs

We introduced uniform DEC-POMDPs in Section 4.6.4 to account for problem settings where all agents and their policies are strictly identical. We also mentioned that such a property could not help solving DEC-POMDPs via dynamic programming. The situation is entirely different in the case of a search algorithm. We have seen that expanding a n-agent joint policy at depth t means creating a total of $(\mathcal{A}^{\Omega^t})^n$ new child nodes. In the case of a uniform DEC-POMDP however, most of these nodes are irrelevant, since they would lead to different policies for each agent. The number of policies that actually needs to be considered therefore is much smaller. In fact, the number of joint policies where all individual agents' policies are identical equals the number of individual policies for a single agent alone. Therefore, expanding a joint policy in the case of a uniform DEC-POMDP only leads to a total of $\mathcal{A}^{\Omega^t}$ new child nodes. Solving uniform decentralized POMDPs using MAA* can be done by replacing statement 6 of Algorithm 31 with the one shown in Algorithm 34. Establishing an algorithm that solves uniform DEC-

Algorithm 34 Multi-agent A* for uniform DEC-POMDPs - `UniformMAA*()`

6 : $\quad \boldsymbol{q}^{*'} \leftarrow$ `ExpandUniform(`$\boldsymbol{q}^*$`)`

POMDPs with the same complexity than single-agent POMDPs can also be seen as a proof that the complexity for this subclass of problems is lower than for the general DEC-POMDP. Indeed, we state the following theorem :

Theorem 7.2.3. *Solving a finite-horizon uniform DEC-POMDP is PSPACE-complete. It is equivalent to solving a finite-horizon POMDP*

Proof. Consider MAA* for uniform DEC-POMDPs as given in Algorithms 31 and 34. □

7.3 Infinite-horizon Heuristic Search

Extending heuristic search to infinite-horizon problems has already been applied to POMDPs [MKKC99]. This was the primal motivation for extending MAA* in a similar way, and the following section introduces a best-first search algorithm for infinite-horizon DEC-POMDPs that makes the connection between infinite-horizon POMDP search methods, and the finite-horizon DEC-POMDP search algorithm MAA*. This work was first presented in [SC05].

7.3.1 Infinite-horizon Policies

We have noted earlier that infinite-horizon POMDPs cannot be solved optimally without a prior definition of a policy model. The same is true for infinite-horizon DEC-POMDPs, and we have already introduced a widely accepted class of such policies in Section 3.2.4 and in Section 6.3.1, namely finite state controllers. In the following, we will use *deterministic finite state controllers* to establish an infinite-horizon heuristic search algorithm for DEC-POMDP.

Definition 7.3.1 (Deterministic Finite State Controller). *A* deterministic finite state controller *(FSC) for agent i is a tuple $\langle \mathcal{N}_i, \mathcal{A}_i, \Omega_i, \psi_i, \eta_i \rangle$, where*
- *$\mathcal{N}_i$ is a finite set of controller nodes,*
- *$\mathcal{A}_i$ is a finite set of actions,*
- *Ω_i is a finite set of observations,*
- *$\psi_i : \mathcal{N}_i \rightarrow \mathcal{A}_i$ is a deterministic action selection function,*
- *$\eta_i : \mathcal{N}_i \times \mathcal{A}_i \times \Omega_i \rightarrow \mathcal{N}_i$ is a deterministic node transition function.*

We recall that an example of a 3-node FSC is given in Figure 3.11. The execution of each controller has to be be started from a well defined initial node $n_i^0 \in \mathcal{N}_i$. The goal is then to find a joint policy of finite state controllers, such that their concurrent execution, starting from controller nodes $\boldsymbol{n}^0 = \langle n_1^0, \ldots, n_n^0 \rangle$ maximizes the expected amount of reward. As stated earlier, finite memory controllers are naturally limited in treating infinite horizon problems, and increasing the controller size will in general lead to higher expected rewards. We therefore state our optimization criterion as finding the best joint policy *for a given controller size.*

7.3.2 Partially Defined Finite State Controllers

We have seen in a previous section that forward search in the space of joint policies can be considered as an incremental construction of an optimal policy based on evaluations of only partially completed policy stubs. In each step, the most promising stub is selected and further developed, hence the best-first approach. We will now apply the same concept to policies based on finite state controllers. Note that this is in contrast with the dynamic programming approach presented in Section 6.3.1 : while the dynamic programming approach tends to optimize the parameters of a fully specified joint policy, the heuristic search approach tends to incrementally construct an optimal joint policy by adding, with each iteration, new parameters to a partially defined policy. We will now detail what we understand by partially defined finite state controllers in the context of infinite-horizon DEC-POMDPs, and how a heuristic function can be established to evaluate these partially defined joint policies.

We say that a finite state controller is *partially defined*, if its action selection function, or its node transition function, or both, are only partially defined. If the transition functions are tables, then a partially defined controller can be represented as a table with blanks. Search in the space of partially defined joint policies then proceeds similar to the previous MAA* algorithm, namely by selecting the current best partially developed joint policy, and constraining it one step further. Figure 7.4 gives an example of how a finite state controller can iteratively be constrained. A section of the infinite-horizon variant of the multi-agent search tree is shown in Figure 7.5. We can immediately determine the depth of the search tree, if, at each step of the algorithm, all n policies are simultaneously constrained : if we assume $|N|$ nodes per controller, then there are $|N|$ actions (one per node) and $|N||\Omega|$ successor nodes ($|\Omega|$ successor nodes per node) that have to be determined. This makes a total depth of the search tree of $|N| + |N||\Omega|$.

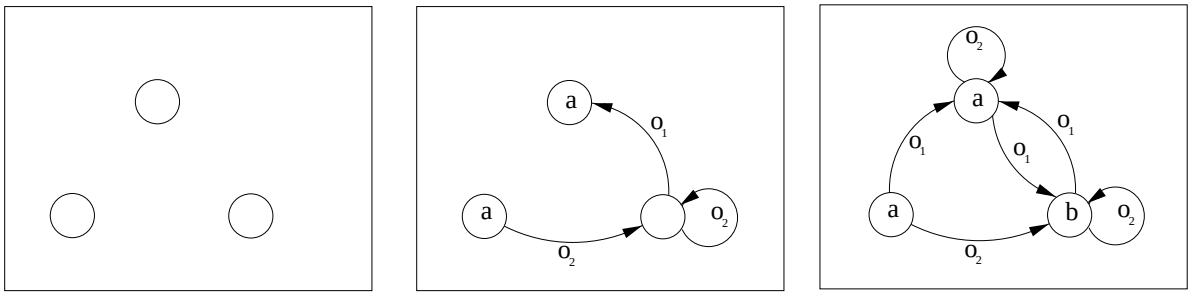

FIG. 7.4 – Three different stages of constraining a three-node finite state controller. Left : an empty FSC. Center : a partially defined FSC. Right : a fully specified FSC (for an environment with 2 observations o_1 and o_2).

We recall that $\boldsymbol{q}^N$ denotes a joint policy of FSCs with N nodes each. For a completely defined joint policy $\boldsymbol{q}^N$, we set $V(s_0, \boldsymbol{q}^N)$ as the value of executing $\boldsymbol{q}^N$ in s_0, which is nothing more than the expectation introduced in (3.5). The maximization problem thus becomes :

$$\boldsymbol{q}^* = \arg \max_{\boldsymbol{q}^N \in \boldsymbol{Q}^N} V(s_0, \boldsymbol{q}^N) \tag{7.11}$$

Evaluating joint policies, and estimating partially defined policy stubs, can be done using the model parameters $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{O}$ of the DEC-POMDP. We will describe this in more detail in the following sections.

7.3.3 Evaluating Finite State Controllers

It has been pointed out by Sondik and later by Hansen [Han97] that evaluating a policy represented as a finite state controller consists in solving a system of linear equations. In fact, it resembles the policy evaluation step in policy iteration. The idea is to establish a value function over the *cross-product* between the underlying system states and the controller nodes $\mathcal{S} \times N$. The value $V(s, n, q)$ then indicates the expected amount of accumulated reward when the execution of the FSC q is started in node n with underlying system state s. Since the controller is fully specified, establishing this value function is equivalent to solving a Markov chain :

$$V(s, n, q) = R(s, \psi(s)) + \gamma \left[\sum_{s', o} P(s', o | s, \psi(s)) V(s', \eta(s, o, q)) \right] \tag{7.12}$$

Extending this evaluation technique to multi-agent joint policies is straightforward. Indeed, it is sufficient to consider the cross-product between system states and joint controller nodes $\mathcal{S} \times N_1 \times ... \times N_n$, such that the value $V(s, \boldsymbol{n})$ indicates the expected amount of reward for the team when execution is started in $\boldsymbol{n}$ with underlying system state s. It remains to be shown how this technique can be extended to partially defined policy stubs.

7.3.4 The Heuristic Function

We recall that a FSC i is defined by a set of nodes $\mathcal{N}_i$, and two functions $\psi_i : \mathcal{N}_i \rightarrow \mathcal{A}_i$ and $\eta_i : \mathcal{N}_i \times \mathcal{A}_i \times \Omega_i \rightarrow \mathcal{N}_i$. A *policy stub* is a joint policy which contains only partially defined FSCs.

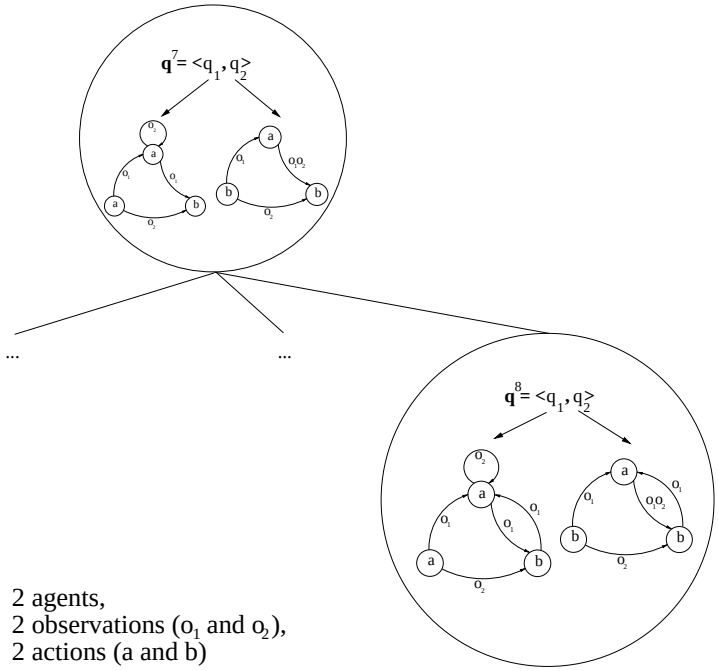

FIG. 7.5 – A section of the multi-agent best-first search tree, showing a partially defined joint policy for 2 agents, and one of its stronger constrained child policies.

Obviously, any partially defined FSC can easily be completed at random by assigning actions and successor nodes at those points where ψ and η are not constrained. This would lead to a lower bound for the possible best completion of the controller. The crucial step however consists in estimating all possible completions of the policy stub in order to determine an upper bound for its best completion. This upper bound then constitutes the heuristic function that determines whether to expand the corresponding leaf node of the search tree or not. We will show that such an upper bound estimate can indeed be established. The basic idea is to extend the cross-product Markov chain to include the decisions about which action (resp. successor node) to chose at those entries where ψ (resp. η) are not yet defined. Such a cross-product MDP has been established in [MKKC99]. We now extend their result to the multi-agent case.

We introduce two mappings that specify the current constrainment of a FSC. The mapping $\Lambda_i(n)$ returns the set of possible actions available in controller node n. If ψ is defined in n, then $\Lambda_i(n) = \{\psi(n)\}$, otherwise $\Lambda_i(n)$ returns the action set of agent i. The mapping $\Pi_i(n, o)$ is defined in a similar way and returns the set of possible successor nodes for observation o in node n. The two functions are thus defined as follows :

$$\Lambda_i(n) = \begin{cases} \{\psi_i(n)\}, & \text{if } \psi_i \text{ is defined in } n \\ \mathcal{A}_i, & \text{otherwise} \end{cases}$$

$$\Pi_i(n, o) = \begin{cases} \{\eta_i(n, o)\}, & \text{if } \eta_i \text{ is defined for } n \text{ and } o \\ \mathcal{N}_i, & \text{otherwise} \end{cases}$$

We then define the multi-controller extensions $\Lambda(\mathbf{n}) = \langle \Lambda_1(n), \ldots, \Lambda_n(n) \rangle$ and $\Pi(\mathbf{n}, \mathbf{o}) = \langle \Pi_1(n, o), \ldots, \Pi_n(n, o) \rangle$ in a similar way. The equation (7.12) can now be extended to

partially defined controllers by maximizing ψ over Λ, and η over Π :

$$V(s,n,q) = \max_{a\in\Lambda(n)} \left\{ R(s,a) + \gamma \left[\sum_{s',o} P(s',o|s,a) \max_{n'\in\Pi(n,o)} V(s',n',q) \right] \right\} \tag{7.13}$$

The interpretation of this equation is the following : whenever the finite state controller q is undefined for the current node or the current node-observation pair, a maximization over all possible actions and all possible successor nodes is allowed. This maximization explicitly depends both on the underlying state, and the successor state after the transition. Such information however is not given during execution, since the system is assumed to be partially observable. The above process thus simulates a partial oracle POMDP, where the real system state can only sometimes be revealed. This can be interpreted as an equivalent upper bound evaluation as the MDP heuristic in finite-horizon MAA*.

We now extend the notion of a cross-product between a policy and a POMDP to the multi-agent case :

Definition 7.3.2 (Multi-agent cross-product MDP). *Given a DEC-POMDP* $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$ *and a joint policy* $\boldsymbol{q} = \langle \{\mathcal{N}_i\}, \{\alpha_i\}, \{\eta_i\} \rangle$ *of - possibly partially defined - finite state controllers, we define the* multi-agent cross-product MDP $\langle \overline{\mathcal{S}}, \overline{\mathcal{A}}, \overline{\mathcal{T}}, \overline{\mathcal{R}} \rangle$ *as follows :*

- $\overline{\mathcal{S}} = (\underset{i}{\times} \mathcal{N}_i) \times \mathcal{S}$
- $\overline{\mathcal{A}} = \underset{i}{\times} (\mathcal{A}_i \times \mathcal{N}_i^{\Omega_i})$
- $\overline{\mathcal{T}}((s,\mathbf{n}),(\mathbf{a},\eta^{\mathbf{n}}),(s',\mathbf{n}')) = P(s,\mathbf{a},s') \sum_{\substack{\mathbf{o}\in\times\boldsymbol{\Omega} \\ s.t.\ \eta^{\mathbf{n}}(\mathbf{o})=\mathbf{n}'}} O(s,\mathbf{a},\mathbf{o},s')$
- $\overline{\mathcal{R}}((s,\mathbf{n}),(\mathbf{a},\eta^{\mathbf{n}})) = R(s,\mathbf{a})$

where $\eta^{\mathbf{n}}$ *is a mapping that - given a vector of nodes* $\mathbf{n}$ *- determines a vector of successor nodes* $\mathbf{n}'$ *for each vector of observations* $\mathbf{o}$*,* $\eta^{\mathbf{n}} : \boldsymbol{\Omega} \to \mathcal{N}$*.*

Solving the cross-product MDP can be done using common dynamic programming techniques, leading to a value function over the augmented state space $\overline{\mathcal{S}}$ and the following fix point :

$$\overline{V}(s,\boldsymbol{n},\boldsymbol{q}) = \max_{\boldsymbol{a}\in\Lambda(\boldsymbol{n})} \left\{ R(s,\boldsymbol{a}) + \gamma \left[\sum_{s',\boldsymbol{o}} P(s',\boldsymbol{o}|s,\boldsymbol{a}) \max_{\boldsymbol{n'}\in\Pi(\boldsymbol{n},\boldsymbol{o})} \overline{V}(s',\boldsymbol{n'},\boldsymbol{q}) \right] \right\} \tag{7.14}$$

This value function is the multi-agent extension of the one given in [MKKC99]. It constitutes the MDP heuristic for the infinite-horizon MAA* algorithm, since it overestimates the value of the optimal completion of joint policy $\boldsymbol{q}$:

Lemma 7.3.1. *For any joint policy* $\boldsymbol{q'}$ *that can be obtained from* $\boldsymbol{q}$ *by adding further constraints on* Λ *or* Π*,* $\overline{V}(s,\boldsymbol{n},\boldsymbol{q}) \geq \overline{V}(s,\boldsymbol{n},\boldsymbol{q'}), \quad (\forall s,\boldsymbol{n})$*.*

Proof. The lemma states that $\overline{V}(*,*,\boldsymbol{q})$ is indeed an upper bound for all joint policies that might result from joint policy $\boldsymbol{q}$ by further constraining it. This is true, since constraining Λ or Π will simply result in reducing the set of options under the max-operators $\max_{\mathbf{a}\in\Lambda(\mathbf{n})}$ and $\max_{\mathbf{n}'\in\Pi(\mathbf{n},\mathbf{o})}$ in (7.14), and the value function can therefore never increase. □

If the joint policy of FSCs is completely defined, the cross-product MDP degenerates to a simple Markov chain, and the upper bound coincides with the true value of the policy. We can finally evaluate an upper bound for the value of any partially defined joint policy and start state s_0 :

$$\overline{V}(s_0, \boldsymbol{q}) = \max_{\mathbf{n} \in \mathcal{N}} \overline{V}(s_0, \mathbf{n}, \boldsymbol{q}) \tag{7.15}$$

7.3.5 Infinite-horizon MAA*

The infinite-horizon MAA* algorithm that optimizes joint policies of deterministic finite state controllers is summarized in Algorithm 35. As opposed to classical A* search techniques, where the evaluation function can explicitly be separated into two parts, the heuristic for evaluating partially defined FSCs is more complex. We now state the main theorem, namely that infinite-horizon MAA* indeed finds an optimal joint policy for the given controller size.

Theorem 7.3.1. *The heuristic best-first search algorithm in* Algorithm 35 *is complete and returns the optimal solution for the given controller size.*

Proof. The search process will eventually terminate in the worst case after enumerating all possible joint policies, which means after constructing the complete search tree. The leaf node with the highest value then contains an optimal solution to the problem. If the search terminates earlier and returns a joint policy $\boldsymbol{q}$, we can guarantee by the "best-first" property that no other active leaf node presents a higher evaluation. Since the evaluation function itself constitutes an upper bound for the value of any further constrained joint policy (see Lemma 7.3.1), we know that all unvisited child nodes will present values that fall below this bounded. This excludes the existence of any joint policy with a higher value, and thus guarantees the optimality of the solution. □

7.4 Conclusion

The heuristic search algorithms for solving decentralized POMDPs presented in this work are entirely novel. This can be seen as a response to the question raised by Hansen et al. in [HBZ04], namely whether forward search is possible for multi-agent problems. We successfully established a finite-horizon and an infinite-horizon version of the new multi-agent A* algorithm, based on previous work by Washington, who showed that the MDP value function could be used as a heuristic for policy search in POMDPs, and Hansen and Meuleau et al., who established an easily computable upper bound for the expected value of a partially defined finite state controller.

MAA* search has some advantages over the dynamic programming algorithms presented earlier, especially concerning its memory requirements, and its ability to speed up the computation in the case of uniform DEC-POMDPs. Future work should include extending the search for optimal joint finite state controllers, extending earlier work about policy search for POMDPs [Han98] : instead of constraining the size of the policy at the beginning of the search process, it is equally possible to add new controller nodes at each iteration, and to rearrange the node transition function while the search proceeds. We are confident that these extension can help improving the algorithms introduced in this chapter. They are summarized in Figure 7.6.

Algorithm 35 Infinite-horizon MAA* - `InfiniteHorizonMAA*()`

Require: A DEC-POMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R} \rangle$, a start state s_0, and a policy size N
Ensure: An optimal joint policy

Initialize open list D with an unconstrained joint policy of size N
$\boldsymbol{q}_{temp} \leftarrow \arg\max_{\boldsymbol{q} \in D} \overline{V}(s_0, \boldsymbol{q})$
repeat
  /* Expand current best solution */
  Select $\boldsymbol{q}^* \leftarrow \arg\max_{\boldsymbol{q} \in D} \overline{V}(s_0, \boldsymbol{q})$
  $\boldsymbol{q}^{*'} \leftarrow$ `Constrain(`$\boldsymbol{q}^*$`)`
  /* Check for new suboptimal solution
  if $\overline{V}(s_0, \boldsymbol{q}_{temp}) < \overline{V}(s_0, \boldsymbol{q}^{*'})$ **then**
    $\boldsymbol{q}_{temp} \leftarrow \boldsymbol{q}^{*'}$
    Output $\boldsymbol{q}^{*'}$
    for all $\boldsymbol{q} \in D$ **do**
      if $\overline{V}(s_0, \boldsymbol{q}) \leq \overline{V}(s_0, \boldsymbol{q}^{*'})$ **then**
        $D \leftarrow D \setminus \{\boldsymbol{q}\}$
      end if
    end for
  end if
  /* Add new joint policy to open list */
  $D \leftarrow D \bigcup \{\boldsymbol{q}^{*'}\}$
  if $\boldsymbol{q}^*$ is fully constrained **then**
    $D \leftarrow D \setminus \{\boldsymbol{q}^*\}$
  end if
until $\exists \boldsymbol{q}^* \in D$ complete such that $\forall \boldsymbol{q} \in D : \overline{V}(s_0, \boldsymbol{q}) \leq \overline{V}(s_0, \boldsymbol{q}^*) = V(s_0, \boldsymbol{q}^*)$

	MAA*, [SCZ05]
Model	DEC-POMDP
Assumptions	Single start state
Horizon	Finite
Solution	Optimal policies
Collaboration	Cooperative
Description	A* on joint policies based on decision trees.

	Infinite-horizon MAA*, [SC05]
Model	DEC-POMDP
Assumptions	Single start state
Horizon	Infinite
Solution	Approximation (based on controller size)
Collaboration	Cooperative
Description	A* on joint policies based on deterministic finite automata.

FIG. 7.6 – Solving DEC-POMDPs using heuristic search.

Chapitre 8

Solving DEC-POMDPs using Reinforcement Learning

8.1 Introduction to Reinforcement Learning

Learning policies by reinforcements is a technique that tends to imitate the behavior of animals, and is based on trying several alternatives, and choosing the one that produced the best output based on a feedback signal, commonly called reward. The idea of computer-based reinforcement learning is to establish a value function over all possible outcomes, and to approximate the optimal action-outcome mapping by testing all alternatives. The difficulty lies in the fact that a negative immediate reward can lead to a configuration with very high rewards in the long run.

Reinforcement learning means exploring an unknown environment while at the same time optimizing a policy that maximize the collection of rewards. Exploration and learning happens on-line, which means in particular that no central controller can help the agents in coordinating their behavior. It also means that the agents have access to a reward immediately after each state transition. This was usually not the case in the previous sections, where the reward was a global criterion that could not be accessed on a local scale. This difference is interesting in the case of partially observable environments, since the reward signal could be used to gain additional information about the current state.

8.2 Multi-agent Q-learning

We will be using the popular Q-learning algorithm, introduced by Watkins [Wat89, WD92], as a basis for multi-agent learning. We will revise it first in its original form, before introducing possible multi-agent extensions, and discussing several issues related to the distribution of the learning process.

8.2.1 Q-learning

The Q-learning approach consists in incrementally approaching the optimal state-action value function of the system by testing actions and observing their outcomes. For each such state-action-outcome trial, the associated Q-value is updated as follows :

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a') \right] \tag{8.1}$$

We will now discuss how this update can be extended to cases where multiple agents all execute their actions together.

8.2.2 Issues in Multi-agent Q-learning

Extending Q-learning to decentralized systems is very appealing, because of the simplicity of the learning rule. It is however not straightforward. Tan was maybe the first one to prove by example that applying single-agent Q-learning to each agent separately does not always lead to optimal solutions in multi-agent domains [Tan97].

Partial Observability The basic assumption of the Q-update is that all information related to a state transition, namely the current state, the action, the next state, and the reward signal, is known by the agent. One can argue whether this is still the case in a distributed environment : does each agent know which actions have been executed by its teammates ? Does it has knowledge about the reward perceived by any of the other agents ? Even assuming this to be true, the learning algorithms and exploration strategies are usually still hidden. We argue that multi-agent reinforcement learning scenarios are inherently partially observable.

Stationary of the Environment Furthermore, standard Q-learning is based on the assumption that the environment, which might be stochastic, remains stationary. In a multi-agent setting, where agents have to be considered as being part of the environment, this is not true anymore, since all agents are learning at the same time.

Coordination Also, even if the optimal joint actions were known be everyone, there usually remains a problem of coordination. As shown by Boutilier in [Bou99], the knowledge about the existence of the optimal joint actions $\langle a, a\rangle$ and $\langle b, b\rangle$ could still lead to a situation where agent 1 choses to execute local action a, assuming that agent 2 would do the same, and agent 2 choses action b with the assumption that agent 1 does the same. The joint action $\langle a, b\rangle$ however might be very suboptimal.

The Multi-agent Exploration/Exploitation Dilemma As observed by Chalkiadakis and Boutilier in [CB03], there is a generalized exploration/exploitation tradeoff in multi-agent reinforcement learning. It consists in the knowledge one has about the current strategies of the other agents. An agent can either exploit this knowledge and act accordingly, or explore the strategy space of the other agents. Note however that this choice is limited : exploiting current knowledge about other agents' strategies can become useless if those strategies have already been altered during learning.

8.2.3 The Mutual Notification Algorithm

We introduced the mutual notification algorithm in [SC04c] as an attempt to extend Q-learning to multi-agent systems, in a way to guarantee the optimality of the Q-function while keeping the value function update as decentralized as possible. We assume each agent to be an *independent learner* in the sense defined by [CB98]. Independent learners do not consider the actions of their teammates directly, but store only Q-values for their own state-action pairs. This reduces the problem complexity of the state action space, but rises serious problems of coordination as elucidated by Boutilier in [Bou99]. We focus on achieving this coordination by allowing explicit communication between agents, while keeping the amount of messages needed for the algorithm

to converge as low as possible. In order to distribute the learning process, we assume that the reward function is decomposable (see Section 4.6.2). We also assume that the current state is fully observable, and that the transition function is deterministic. We think of applications such as robotic soccer, where a team of robots has to push a ball into a goal. Each robot can fully observe the scene, but local power consumption constitutes a hidden reward at runtime. The formal model we will be using is a *distributed MMDP* :

Definition 8.2.1 (Distributed MMDP). *Given an MMDP, the associated* distributed MMDP *is given as follows :*

- $\mathcal{S}$ *is the set of states,* $s \in \mathcal{S}$*,*
- $\mathcal{A}_i$ *is the set of actions available to agent* i*,*
- $\mathcal{T}(s, \langle a_1, a_2, \ldots, a_n \rangle, s')$ *is the global transition function of the system,*
- $r_i(s, a_i)$ *are the individual reward functions,*
- $\pi_i(s)$ *are the local policies,*
- $q_i(s, a_i)$ *are the local Q-functions,*
- $v_i(s) = q_i(s, \pi_i(s))$ *are the local value functions.*

Each agent i holds a local Q-function $q_i(s, a_i)$, a local policy $\pi_i(s)$, and a local value function $v_i(s) = \pi_i(s, \pi_i(s))$. Note that the local value function does not verify the equality given in (3.10). Each agent has furthermore access to its own local rewards according to its reward function $r_i(s, a_i)$. We have noticed earlier that a fully observable multi-agent MDP can be reduced to a MMDP if the control is centralized. It is then possible to define a value function over joint actions, and to apply single-agent Q-learning. Our goal is to compute a set of policies that achieve the same performance as a centralized MMDP controller. We will call such a solution *MMDP-optimal*.

The mutual notification algorithm works as follows. Agents compute local Q-functions, but do not perform an update as long as the Q-function does not increase. We introduce $\underline{q}_i(s, a_i)$ as the value of a *possible* Q-update. It is calculated as for Q-learning in deterministic environments :

$$\underline{q}_i(s, a_i) = r_i(s, a_i) + \gamma v_i(s') \tag{8.2}$$

Actions are executed synchronously, and the system proceeds from state s to state s' as given by the transition function. Each agent is then able to compare $\underline{q}_i(s, a_i)$ with the value of its current policy $q_i(s, \pi_i(s))$, which means test if the new value represents a local improvement or not :

$$\underline{q}_i(s, a_i) \overset{?}{>} q_i(s, \pi_i(s)) = v_i(s) \tag{8.3}$$

The expected local gain for agent i then is

$$\Delta \underline{v}_i(s) = \underline{q}_i(s, a_i) - v_i(s) \tag{8.4}$$

and the accumulated gain for the whole system is

$$\Delta \underline{V}(s) = \sum_i \Big[\underline{q}_i(s, a_i) - v_i(s) \Big] \tag{8.5}$$

Since a global improvement of the value function implies at least one local improvement of some agents' value function, we can state that $(\Delta \underline{V}(s) > 0)$ *iif* $(\exists i)(\Delta \underline{v}_i(s) > 0)$. Therefore, as soon as test (8.3) is valid for an agent, it notifies all other agents by communicating its expected

improvement $\Delta\underline{v}_i(s)$. If notification has occurred, all other agents reply by sending their local improvements, which allows every agent to calculate the global gain (8.5), and to determine whether a global improvement is possible or not. If this is the case, then each agent updates its local value function and policy :

$$\begin{cases} q_i(s,a_i) & \leftarrow & \underline{q}_i(s,a_i) \\ \pi_i(s) & \leftarrow & a_i \\ v_i(s) & \leftarrow & q_i(s,a_i) \end{cases} \tag{8.6}$$

If no global improvement is possible, no update is done by any agent. This algorithm converges to the MMDP-optimal policy and value function, if the initial Q-function is strictly bounded by the optimal Q-function, for example by initializing $(\forall i, s, a_i)\ q_i(s,a_i) \leftarrow -\infty$. It is given in Algorithm 36. We can prove the convergence to a MMDP-optimal solution by constructing a

Algorithm 36 The mutual notification algorithm - `MutualNotification()`

Require: A distributed MMDP $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R} \rangle$
Ensure: An MMDP-optimal joint policy

```
while learning do
  for all agents i do
    Execute an action a_i according to exploration policy
    q_i(s,a_i) ← r_i(s,a_i) + γ v_i(s')
    Δv_i(s) ← q_i(s,a_i) − v_i(s)
    if Δv_i(s) > 0 or message received then
      Broadcast Δv_i(s)
    end if
    if communication has occurred then
      if (Σ_{j=1}^{n} Δv_j(s) > 0) then
        q_i(s,a_i) ← q_i(s,a_i)
        π_i(s) ← a_i
        v_i(s) ← q_i(s,a_i)
      end if
    end if
  end for
end while
```

modified MMDP that simulates the distributed process. We will define the following functions : Q, V, and Π are the Q-function, value function, and policy of the centralized MMDP. We will set $\tilde{Q}$, $\tilde{V}$, and $\tilde{\Pi}$ as the Q-function, value function, and policy of the modified process, and define the following update rules :

$$\begin{array}{rcl} \underline{\tilde{Q}}(s,\boldsymbol{a}) & \leftarrow & R(s,\boldsymbol{a}) + \gamma \tilde{V}(s') \\ \tilde{Q}(s,\boldsymbol{a}) & \leftarrow & \begin{cases} \underline{\tilde{Q}}(s,\boldsymbol{a}) & \text{if } \underline{\tilde{Q}}(s,\boldsymbol{a}) > \tilde{V}(s) \\ \bar{\tilde{Q}}(s,\boldsymbol{a}) & \text{otherwise} \end{cases} \end{array} \tag{8.7}$$

We now have three different processes, the centralized MMDP process, the modified centralized MMDP process, and the distributed process. We now show that the modified centralized process converges to the same value function and policy than the original MMDP.

Lemma 8.2.1. *The centralized MMDP process and the modified centralized MMDP process converge to the same values functions and policies.*

Proof. We actually show that if $\tilde{Q}, \tilde{V}, \tilde{\Pi}$ and Q, V, Π are initialized with the same values, then the series of the state value functions V and $\tilde{V}$, and policies Π and $\tilde{\Pi}$ are identical when the same sequence of actions is used. This can be done by induction over the iteration step. The induction holds at the beginning by initializing all functions with the same values. According to the update rule in (8.7), we then have to distinguish two cases :

a) $\underline{\tilde{Q}}(s, \boldsymbol{a}) > \tilde{V}(s)$

This means that the update in (8.7) is done. Since the update is always done in the original centralized MMDP, and since by induction hypothesis the value functions and policies where identical during the previous iteration, they are still identical after the update.

b) $\underline{\tilde{Q}}(s, \boldsymbol{a}) \leq \tilde{V}(s)$

In this case the update in (8.7) is not done, and nothing is changed in the modified process. In the original MMDP process, an update of the Q-function is done. However, since the joint action was sub-optimal, the value function and the policy remain unchanged for the original process as well. □

We then show that the distributed process converges to the same policy than the modified centralized process. This proves that the decentralized process is MMDP-optimal.

Lemma 8.2.2. *The modified centralized MMDP process and the decentralized process converge to the same values functions and policies.*

Proof. We actually show that if $\tilde{Q}, \tilde{V}, \tilde{\Pi}$ and q_i, v_i, π_i are initialized with the same values, then the series of the state value functions $\tilde{V}$ and v_i, and policies $\tilde{\Pi}$ and π_i are identical when the same sequence of actions is used. This can again be done by induction over the iteration step. The induction holds at the beginning by initializing all functions with the same values. We then show that the test $\Delta\underline{V}(s) > 0$ is equivalent to the test in (8.7) :

$$\begin{aligned}
& & \Delta\underline{V}(s) &> 0 \\
\Leftrightarrow & \sum_{i=1}^{n} \left[\underline{q_i}(s, a_i) - v_i(s)\right] &> 0 \\
\Leftrightarrow & \sum_{i=1}^{n} \underline{q_i}(s, a_i) &> \sum_{i=1}^{n} v_i(s) \\
\Leftrightarrow & \sum_{i=1}^{n} \left[r_i(s, a_i) + \gamma v_i(s')\right] &> \sum_{i=1}^{n} v_i(s) \\
\Leftrightarrow & \underline{\tilde{Q}}(s, \boldsymbol{a}) &> \tilde{V}(s)
\end{aligned}$$

The last line is is true due to the induction hypothesis for the value function of the previous iteration. □

The two lemmas prove the convergence of the mutual notification algorithm in the case of fully observable environments.

Hierarchical Communication Strategies The basic mutual notification algorithm can be extended in several ways. The flat communication act for example can be replaced by a hierarchical approach, thus reducing considerably the amount of communication. Instead of communicating each one with each other, agents can be regrouped in *cliques*, and first determine whether a local clique improvement has been occurred. If this is the case, a representative of the clique can then communicate the cliques' improvement to the representatives of the other cliques, and so on. At the last level, a single agent finally determines whether a global improvement has been occurred or not, and communicates this result backwards. Using a hierarchical mutual notification policy the complexity of the communication overhead can drop from polynomial $O(n^2)$ to linear $O(n)$ in the number of agents.

Threshold Communication Models The mutual notification algorithm uses a fixed communication strategy : As soon as a local improvement is detected, it is communicated to the teammates. To allow a more flexible learning strategy, we introduce a communication threshold δ, and restrict communication to those cases where the local improvement exceeds this threshold $\Delta \underline{v}_i > \delta$. This learning will also converge, and we can identify an iteration step time T_s where the agents policy will become stationary. The policy π_{T_s} is an approximation of the optimal policy, and we can bound the error of π_{T_s} with respect to π^* for a given threshold δ.

The maximal local error for horizon 1 in a state s is $n\delta$, where n is the number of agents - in the worst case, each agent could improve its local value function by δ, but does not communicate this improvement. The maximal error for the infinite horizon is given by

$$E_{T_s} = \sum_{k=0}^{\infty} (n\delta)\gamma^k \tag{8.8}$$

where $\gamma < 1$ is the discounting factor from Q-learning. In order to get an ϵ-approximation of the optimal value function, we calculate for δ

$$|V_{T_s}(s) - V^*(s)| < \epsilon \quad \Rightarrow \quad \sum_{k=0}^{\infty} (n\delta)\gamma^k < \epsilon \tag{8.9}$$

$$\Rightarrow \quad \delta < \frac{(1-\gamma)\epsilon}{n} \tag{8.10}$$

The threshold can furthermore be time dependent and decreasing δ_t. This makes it possible to focus on the important updates only at the beginning, and to take smaller updates into account as the learning proceeds. One possibility is to define $\delta_0 = R_{max}$ (the highest reward value, since the reward function is bounded), whereby the existence of a value T with $\delta_T < \frac{(1-\gamma)\epsilon}{n}$ has to be guaranteed.

Extensions to Partially Observable Problems Another interesting question is whether mutual notification can be extended to partially observable environments. It is known that optimal policies for partially observable environments are usually memory-based, although some exceptions exist, such as the locally fully observable DEC-MDP for example (see Section 4.6.2). Learning policies that are based on local states via mutual notification might be done as follows : define a mapping I_i that returns, for example for each action a_i and observation (or local state) o_i, the set of possible underlying global states. Then compute *bounds* on a possible update

assuming optimistic and pessimistic state transitions :

$$\underline{q}_i = r_i + \min_{s' \in I_i} \gamma v_i(s') \tag{8.11}$$

$$\overline{q}_i = r_i + \max_{s' \in I_i} \gamma v_i(s') \tag{8.12}$$

In order to determine whether a local improvement has been detected or not, one could define heuristics that compare the bound of the (unknown) next state with the bound of the (unknown) current state. An *optimistic* communication strategy would initiate communication as soon as a local improvement has probability greater than zero :

$$\overline{q}_i \overset{?}{>} \min_{s \in I_i} v_i(s) = \underline{v} \tag{8.13}$$

A *pessimistic* communication strategy would only initiate communication if a local improvement has been detected with certainty :

$$\underline{q}_i \overset{?}{>} \max_{s \in I_i} v_i(s) = \overline{v} \tag{8.14}$$

The above sketch, based on the interval estimation method, is uncomplete, and it is so far unclear how to define value functions over states that are potentially unknown during learning. One might however be able to build an approximative learning algorithm for partially observable multi-agent systems using the principle of mutual notifications.

8.2.4 Emphatic Q-learning

The previous algorithm was developed under the assumption that agents are independent learners, which means that they do not take the behavior of the remaining teammates into account. If this assumption is relaxed, Q-learning can be extended in a different way to multi-agent systems. Indeed, if each agent can observe both the local actions and the local rewards of all of its teammates, it can include this knowledge into the Q-function itself by considering joints actions and joint rewards. In the case of fully cooperative settings with a single reward function, the second assumption is valid by definition.

The idea of emphatic Q-learning is to allow each agent to learn a global Q-function that is the same for all other agents, since all relevant parameters for the Q-update are globally accessible. This includes estimating the value of the next state s', the last term in the Q-update. In single-agent Q-learning, this can be done by optimizing its own action choice. In multi-agent settings however, this includes maximizing the joint action choice. *Empathy* is the ability of an agent to make a virtual decision for another agent by placing itself in its context. It has already been studied in the context of multi-agent systems [CSC02], and allows to maximize the value function of the successor state s' by assuming that all other agents are interested in the same maximization during their own learning process. This can easily be justified by rationality assumptions for cooperative environments. In empathic Q-learning, the update in (3.19) then simply becomes :

$$Q(s, \boldsymbol{a}) \leftarrow (1 - \alpha) Q(s, \boldsymbol{a}) + \alpha \left[R(s, \boldsymbol{a}) + \gamma \max_{\boldsymbol{a'} \in \boldsymbol{\mathcal{A}}} Q(s', \boldsymbol{a'}) \right] \tag{8.15}$$

Note that the exploration of the environment has to be guaranteed in the same way as for single-agent domains. It remains to be shown that empathic Q-learning indeed converges to an optimal solution. This can be assumed, however, because of the proved convergence of a larger class of algorithms, which we will present in the following sections.

8.2.5 Nash Q-learning

The idea of empathic Q-learning has also been extended to non-cooperative scenarios in several ways, and we will present one such idea, Nash Q-learning, introduced by Hu and Wellman [HW03]. By analyzing the update in equation (8.15), one can identify two major aspects that differ when passing from a cooperative to a non-cooperative system. The first one concerns the reward function : in a competitive environment, each agent holds its own reward function, which makes it impossible to integrate the entire set of rewards into a single Q-update. The second one concerns the value of the successor state s' : since each agent tries to maximize only its own payoff, it is no more possible to simply maximize the value over joint actions. As shown by Nash, the optimal joint action for a competitive scenario is an equilibrium strategy, hence the idea behind Nash Q-learning to consider the value of the Nash equilibrium in s' rather than the optimal joint action. Note that the same idea has already been applied by previous techniques (see for example Section 5.3.1, Section 5.3.2, and Section 5.3.3).

Nash Q-learning assumes full observability of actions and rewards. This enables an empathic agent to learn simultaneously its own Q-function and approximate that of all other agents. The approximation is due to the possibly different initialization of the local Q-functions, but vanishes during learning. Each agent thus holds a set of n Q-functions. This enables an empathic agent to compute a Nash equilibrium for the successor state s', assuming a stochastic game structure in s', and with strategy profiles given by the Q-functions of the agents. The update rule of the Nash Q-learning algorithm thus becomes :

$$\forall i : \quad Q_i(s, \boldsymbol{a}) \leftarrow (1-\alpha)Q_i(s, \boldsymbol{a}) + \alpha \left[R_i(s, \boldsymbol{a}) + \gamma Nash Q_i(s')\right] \tag{8.16}$$

We have already mentioned the possibility of multiple Nash equilibria, and the issue of equilibrium selection. Assuming this issue can be addressed in some way during learning, then Nash Q-learning converges to the optimal Nash Q-values of the associated stochastic game :

Theorem 8.2.1. *Nash Q-learning converges to the Nash Q-values of the system.*

Proof. See [HW03]. □

The authors show that the space complexity of the Nash Q-learning algorithm is exponential in the number of agents, because of the need to maintain n Q-functions. The runtime is dominated by the calculation of the Nash equilibria.

8.2.6 Multi-agent Bayesian Reinforcement Learning

As explained earlier, multi-agent reinforcement learning is inherently a partial observable problem, even if all actions and rewards can globally be observed. It is therefore only natural to introduce the belief state concept into multi-agent reinforcement learning. Chalkiadakis and Boutilier define such a belief state as a probability distribution over the space of possible game models P_M, a joint distribution over the possible strategies played by the other agents P_S, the current state of the system s, and the history of the game h :

$$b = (P_M, P_S, s, h) \tag{8.17}$$

They show that this belief state can indeed be updated using Bayes rule [CB03]. It is then possible to establish a *belief state Q-function* $Q(b_i, a_i)$ that can be obtained through the following Bellman

equation

$$Q(b_i, a_i) = \sum_{\boldsymbol{a_{-i}}} P(\boldsymbol{a_{-i}}|b_i) \sum_{s' \in \mathcal{S}} P(s'|\langle a_i, \boldsymbol{a_{-i}}\rangle, b_i) \sum_{R} P(R|\langle a_i, \boldsymbol{a_{-i}}\rangle, b_i) \left[R + \gamma V(b_i')\right] \tag{8.18}$$

with

$$V(b_i) = \max_{a_i \in \mathcal{A}_i} Q(b_i, a_i) \tag{8.19}$$

and where R is the reward value. The important additional information encoded in this belief state Q-function, as argued by Chalkiadakis and Boutilier, is the *expected value of information* of an action. Whereas the classical Q-function is a straightforward evaluation of the action choice in a given state, the belief state Q-function also reflects the impact of the resulting observation on the current belief state.

Establishing a belief state Q-function is a similar challenge than solving the belief state MDP for an associated POMDP (see Definition 3.2.2), and would require solving a high-dimensional continuous system of equations, which can quickly become computationally infeasible. Since most belief states are unreachable or irrelevant given a specific initial belief, the authors propose a myopic estimation of a restricted set of future beliefs in order to approximate the value of the current belief [CB03]. They define a *myopic Q-function* $\tilde{Q}$, similar to the original Q-function given in (8.18), but instead of determining the value of the successor belief state explicitly, a myopic estimation $\tilde{V}$ is used :

$$\tilde{V}(b_i) = \max_{a_i \in \mathcal{A}_i} \int_m \int_{\boldsymbol{\pi_{-i}}} Q(s, a_i|m, \boldsymbol{\pi_{-i}}) P_M(m) P_S(\boldsymbol{\pi_{-i}}) \tag{8.20}$$

The myopic value function is defined "*as the expected value of the optimal action at the current state, assuming a fixed distribution over models* [m] *and strategies* [π]" of the other agents. The term $Q(s, a_i|m, \boldsymbol{\pi_{-i}})$ can easily be evaluated as in standard, fully-observable MDPs. The integration over a possibly infinite set of models however is usually intractable, but the authors suggest to sample from a set of models, and to average their values accordingly.

The Bayesian Q-learning approach constitutes an interesting direction for doing reinforcement learning in multi-agent systems, as it integrates the concepts of belief states, and especially doing on-line belief state updates, into classical Q-learning. It is heavily related to earlier work in single-agent reinforcement learning, particularly addressing the issue of efficient exploration [DFA99].

8.2.7 Other Multi-agent Q-learning Approaches

Early work on two-player reinforcement learning was that of Littman [Lit94], who made the connection between Markov games and reinforcement learning. He also introduced friend-or-foe Q-learning for two-player zero-sum games, and showed how to apply the min-max-principle from game trees to reinforcement learning, under the assumption that the agent is aware of the cooperative of competitive nature of the problem setting [Lit01]. The friend version of the update evaluates the successor state s' by maximizing over joint actions, while the foe version assumes the worst possible action of the adversary and maximizes its own action accordingly :

$$\begin{aligned} \text{friend}: \quad V_i(s') &= \max_{a_i} \max_{a_{-i}} Q_i(s', \langle a_i, a_{-i}\rangle) \\ \text{foe}: \quad V_i(s') &= \max_{P_i \in \Delta\mathcal{A}_i} \min_{a_{-i}} \sum_{a_i \in \mathcal{A}_i} P(a_i) Q_i(s', \langle a_i, a_{-i}\rangle) \end{aligned} \tag{8.21}$$

The Nash Q-learning algorithm presented above has led to a class of related solution concepts. Its convergence proof is based on a very generic theorem which allows the replacement of the Nash-Q-operator in equation (8.16) with other solution concepts for stochastic games. Greenwald and Hall for example replace the Nash equilibrium with four different types of *correlated equilibria*, leading to *utilitarian*, *egalitarian*, *republican*, and *libertarian* correlated Q-learing respectively [GH03]. Veerbeck et al. extended equilibrium based reinforcement learning techniques to a special class of cooperative problems, so called *diagonal games* [VNLP02]. They show that Pareto optimal solutions for a team of agents can be found when using a limited amount of coordination. Lauer and Riedmiller have presented a multi-agent variant of Q-learning, similar to our mutual notification algorithm, in [LR00]. They also applied a Q-learning technique to coordinate agents in electrical power grids [RMS02]. While common reinforcement learning algorithms do not assume the knowledge about the underlying model, that is the transition function, Brafman and Tennenholtz specifically address the issue of model-based coordination [BT03]. Their approach considerably reduces the convergence rate to polynomial complexity, and is guaranteed to reach optimal solutions.

In Figure 8.1, we finally give two examples of cooperative and competitive reinforcement learning problems that have been addressed in the literature, and that can be solved by the algorithms presented so far. A comparative study of the performances of various multi-agent reinforcement learning algorithms can be found in [LLB06].

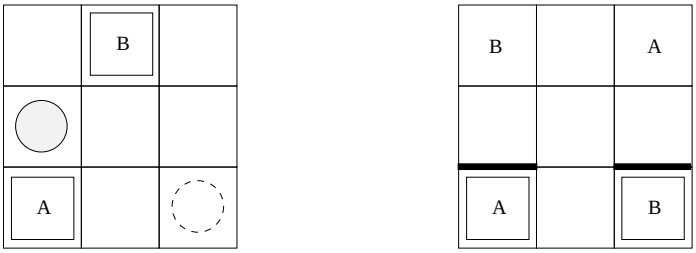

FIG. 8.1 – Two typical multi-agent reinforcement learning scenarios. The first one is a cooperative problem, where two agents have to push a ball on goal. Such a problem can be addressed by the mutual notification algorithm or general empathic Q-learning. - The second one is a competitive scenario, where each agent tries to reach its own goal location, but where the path is partially blocked by two barriers. Such a problem can be addressed by Nash Q-learning, correlated Q-learning, and related techniques.

8.3 Gradient Ascent Techniques

While the previous algorithms were heavily table-based, gradient ascent techniques can be adopted when continuous function have to be optimized. This is for example the case when considering POMDPs. In the following section, we will present a simplified overview of how reinforcement learning using gradient ascent can achieve policy optimization in decentralized partially observable MDPs.

8.3.1 Gradient Ascent

Gradient ascent is a local optimization technique that approaches an extremum of a given function by following its curvature. The general idea is very simple : given a function F on a multi-dimensional domain $\boldsymbol{X}$, the goal of a gradient ascent algorithm is to determine an $\boldsymbol{x}^* \in \boldsymbol{X}$ such that $F(\boldsymbol{x}^*)$ constitutes a maximum of F. The algorithm will start with an initial $\boldsymbol{x}$, evaluate F at $\boldsymbol{x}$, and update $\boldsymbol{x}$ by moving it in the direction of the parameter space $\boldsymbol{X}$ where F increases most. This process is then repeated until $\boldsymbol{x}$ is "sufficiently" close to a maximum of F. The increase of a function is expressed by its gradient, which can be determined by applying the nabla operator, the vector of partial derivatives $\nabla = (\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n})$. The update of $\boldsymbol{x}^i$ can thus be expressed by adding a small fraction of F's gradient :

$$\boldsymbol{x}^{i+1} \leftarrow \boldsymbol{x}^i + \boldsymbol{\alpha}^i \nabla F(\boldsymbol{x}^i) \tag{8.22}$$

The vector $\boldsymbol{\alpha}^i$ represents the size of the increment for each one of the dimensions of $\boldsymbol{X}$. Its value depends in general of the iteration i, thus allowing bigger steps and the beginning of the process, and approaching the optimum more smoothly at the end. A visual interpretation of the gradient ascent method for a one-dimensional function is given in Figure 8.2. Two problems are usually

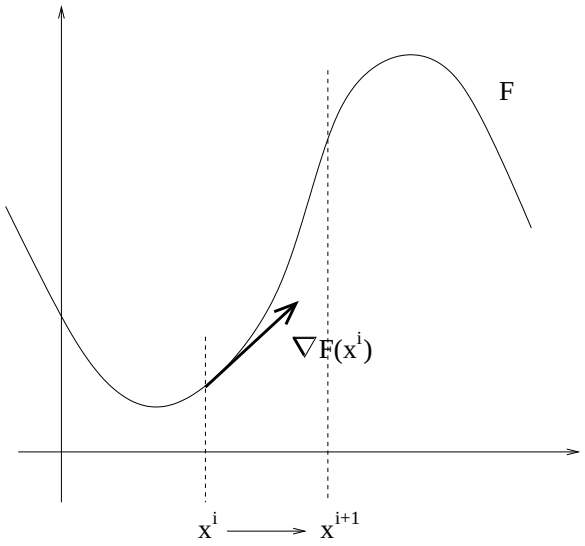

FIG. 8.2 – The general gradient ascent method, which moves a sample point x^i towards the biggest increase of F : based on the gradient $\nabla F(x^i)$, x^i is replaced with a new point x^{i+1} which is hoped to be closer to a maximum of F than x^i.

related to make gradient ascent work. The first one concerns the choice of the step parameter : a small step length usually slows down the convergence of the algorithm, whereas a large increment may lead to an unstable behavior by overshooting the extremum. In general, a variable increment is chosen that decreases with time. The second issue is the stopping criterion, which is usually based on the gradient itself, since a small gradient usually indicates the proximity of an extremum point. A sketch of the generic gradient ascent approach is shown in Algorithm 37.

8.3.2 Parametrization of Policies

Gradient ascent is an appealing technique for optimizing policies when the function itself is not completely known, and has so far been applied for policy search in single-agent and multi-agent

Algorithm 37 Generalized gradient ascent - `GradientAscent()`

Require: A function F and an initial $\boldsymbol{x}^0$
Ensure: An $\boldsymbol{x}$ close to a local maximum of F
  $i \leftarrow 0$
  repeat
    $\boldsymbol{x}^{i+1} \leftarrow \boldsymbol{x}^i + \boldsymbol{\alpha}^i \nabla F(\boldsymbol{x}^i)$
    Update $\boldsymbol{\alpha^i}$
    $i \leftarrow i + 1$
  until $\boldsymbol{x}^i$ is close enough to a local maximum of F

domains [PKMK00, DBC01]. It supposes that the policy can be represented as a function of continuous parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$, and we will briefly mention two possibilities.

Stochastic Memoryless Policies If the policy is a simple mapping from observations to actions $a = \pi(o)$, then a parametrized policy can be obtained by identifying the parameter $\boldsymbol{\theta}(o, a)$ as being the partially observable Q-value for observation-action pair (o, a). A stochastic policy $\pi_{\boldsymbol{\theta}}$ can then be deduced by setting

$$\pi_{\boldsymbol{\theta}}(o, a) = \frac{e^{\boldsymbol{\theta}(o,a)}}{\sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}(o,a')}} \tag{8.23}$$

Finite State Controllers Policies that can be represented as stochastic finite state controllers are implicitly parameter-based. The parameters of the finite state controller, namely its transition and action selection functions, immediately lead to the corresponding parameter vector.

8.3.3 Generalized Multi-agent Gradient Ascent

A general approach for doing value function gradient ascent in partially observable MDPs has been suggested by [PKMK00]. Given a parameter vector $\boldsymbol{\theta}$ and a start state distribution $\boldsymbol{b}_0$, Peshkin et al. suggest to write the value function for infinite horizon POMDPs as follows

$$V(\boldsymbol{\theta}, \boldsymbol{b}_0) = \sum_{t=1}^{\infty} \gamma^t \sum_{h \in H_t} P(h|\boldsymbol{\theta}, \boldsymbol{b}_0) R_t(h) \tag{8.24}$$

where $h \in H_t$ denotes a history of length t, and $R_t(h)$ the reward value produced for history h at time t. The derivative of this function with respect to a parameter θ_i then becomes

$$\frac{\partial V(\boldsymbol{\theta}, \boldsymbol{b}_0)}{\partial \theta_i} = \sum_{t=1}^{\infty} \gamma^t \sum_{h \in H_t} \left(\frac{\partial P(h|\boldsymbol{\theta}, \boldsymbol{b}_0)}{\partial \theta_i} R_t(h) \right) \tag{8.25}$$

which leads to the update $\Delta\theta_i = \alpha_i \frac{\partial V}{\theta_i}$ of parameter θ_i. So far, this algorithm is applicable to single-agent and to multi-agent planning problems. However, computing $P(h|\boldsymbol{\theta}, \boldsymbol{b}_0)$ is usually not possible when reinforcement learning is considered, since the world model is supposed to be unknown. Instead, only samples of histories, collected during the interaction with the environment, can be considered.

For the special case of *factored actions* and *factored controllers*, which means that actions are no longer atomic but consist of several components which are each controlled by a separate controller, the gradient ascent approach can be decentralized. Each controller is then optimized independently. Factored controllers can be considered as being one example of distributed policies for multi-agent systems, where each component represents the action choice of one agent. The question then arises whether distributed optimization of factored controllers leads to a globally optimal policy for the whole multi-agent team. Surprisingly, this is true. Indeed, Peshkin et al. prove that, for factored controllers, distributed gradient ascent is equivalent to joint gradient ascent. They also show that all Nash equilibria are local optima for the gradient ascent approach, although not all local optima are Nash equilibria.

Another general multi-agent gradient ascent approach is the WoLF approach [BV02]. It's *win or learn fast* principle guarantees convergence to a Nash equilibrium by using a special variable learning rate. The idea is to use two different step sizes, one for success and one for failure. The learner then adapts quickly, if it is doing more poorly than expected, and adapts more slowly if it is doing better. The core question is thus to determine adequately if the agent is actually "winning" or not. In their paper, the authors define a winning situation if the agent is preferring his current strategy to that of playing some equilibrium strategy against the other player's current strategy.

8.3.4 Multi-agent Reinforcement Learning by Gradient Ascent : An Example

For the special case of stochastic memoryless policies and the parametrization described above, Buffet shows in his PhD thesis how gradient ascent can be used to allow distributed reinforcement learning in larger multi-agent environments, involving up to 20 agents in a 10×10 grid world [DBC01, Buf03]. He successfully distributed Baxter et al.'s OLPOMDP algorithm [BBW01] to a multi-agent environment. The update of the parameters of the policy, after having executed an action a_t, observed an observation o_t and obtained reward R_t, becomes :

$$\begin{aligned} \boldsymbol{z}_{t+1} &\leftarrow \beta \boldsymbol{z}_t + \frac{\nabla \pi_{\boldsymbol{\theta}}(o_t, a_t)}{\pi_{\boldsymbol{\theta}}(o_t, a_t)} && (8.26) \\ \boldsymbol{\theta}_{t+1} &\leftarrow \boldsymbol{\theta}_t + \alpha R_t \boldsymbol{z}_t && (8.27) \end{aligned}$$

This update can explicitly be computed if the policy is being represented by the parametrization shown above :

$$\frac{\nabla \pi_{\boldsymbol{\theta}}(o_t, a_t)}{\pi_{\boldsymbol{\theta}}(o_t, a_t)} = \nabla \ln \left(\pi_{\boldsymbol{\theta}}(o_t, a_t)\right) = \left(\frac{\partial}{\partial \boldsymbol{\theta}(o_1, a_1)}, \ldots, \frac{\partial}{\partial \boldsymbol{\theta}(o_{|\Omega|}, a_{|\mathcal{A}|})}\right) \ln \left(\pi_{\boldsymbol{\theta}}(o_t, a_t)\right) \quad (8.28)$$

Each component (o_x, a_y) can finally be determined as follows :

$$\frac{\partial \ln(\pi_{\boldsymbol{\theta}}(o_t, a_t))}{\partial \boldsymbol{\theta}(o_x, a_y)} = \frac{\partial \ln \left(\frac{e^{\theta(o_t, a_t)}}{\sum_{a' \in \mathcal{A}} e^{\theta(o_t, a')}}\right)}{\partial \boldsymbol{\theta}(o_x, a_y)} = \delta_{o_t, o_x} \left(\delta_{a_t, a_y} - \boldsymbol{\theta}(o_x, a_y)\right) \quad (8.29)$$

where $\delta_{\cdot,\cdot}$ is the Kronecker symbol. The backups required for the gradient OLPOMDP learning algorithm are thus comparatively easy to execute.

Buffet distributed the above gradient ascent approach, and tested it on a cooperative task where

agents have to push and merge blocks in a grid world (see Figure 8.3). Each agent has a subjective view of the environment (see Section 5.3.1), consisting in the heading of the nearest agent (*North*, *East*, *South*, *West*), and the heading and the distance of the nearest block. This set of observations has the additional property that it is independent of the grid size, thus enabling to scale the approach more easily.

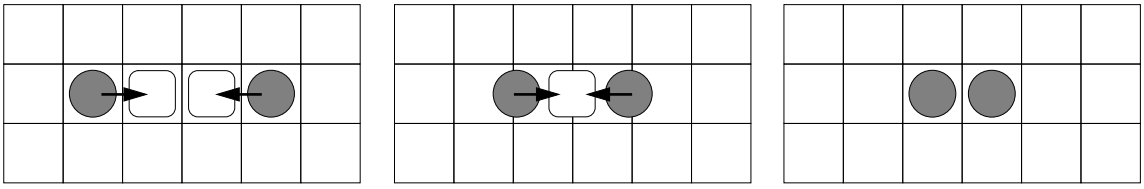

FIG. 8.3 – The pushing blocks scenario as conceived by Buffet. Two agents push two cubes together, which eventually disappear while producing a positive reward. The figure is adapted from [Buf03].

8.4 Critics

Multi-agent reinforcement learning is a far more challenging problem that its planning counterpart. Since planning is centralized, the selection between several Nash equilibria for example can easily be handled. This is not possible anymore during learning, which is really entirely decentralized. As a matter of fact, no complexity results are known for multi-agent reinforcement learning, and no general and optimal algorithms have been identified so far. It is not even established whether globally optimal solutions can be found using reinforcement learning techniques. Shoham et al. recently published a technical report that summarizes some of the problems in research for multi-agent reinforcement learning [SPG03].

8.4.1 The Equilibrium Concept

One issue concerns the choice of the equilibrium concept as a basis for solutions in multi-agent learning. It is known that Nash equilibria are possibly (very) suboptimal in the case of cooperative scenarios. The question then is why equilibria should be used in this case at all. In the case of non-cooperative environments, where algorithms such as Nash Q-learning are supposed to be applicable, it is questionable why agents should be forced to exhibit their local actions and rewards, in order to allow their competitors to estimate their own value functions. It is also not obvious how competing agents can coordinate on a single equilibrium without the help of an external oracle or any other coordination device. In fact, Hart and Mas-Colell argue that the uncoupled dynamics of distributed learning problems make it impossible to guarantee the convergence to a Nash equilibrium in the most general sense [HMC03]. This *"begs the question of why to use learning for coordination at all"* [SPG03].

A different aspect of the equilibrium concept in multi-agent reinforcement learning has been raised by Brafman and Tennenholtz. It does not concern the solution of a distributed learning algorithm, but rather the learning algorithm itself. The authors argue that distributed learning algorithms should be in an equilibrium themselves. This is called an *efficient learning equilibrium* [BT04]. The algorithms are required to be efficient, which means they should converge to the

desired value in polynomial time. They are required to be in an equilibrium, which means that it becomes irrational to deviate unilaterally from such an algorithm after polynomially many steps.

8.4.2 Imperfect Monitoring

Another critical point, which we addressed first in [SC04b], concerns the definition of the reward function. It is commonly agreed on that cooperation among agents is implicitly expressed by the existence of a single reward function, defined over states and joint actions. While this can easily be justified in the context of multi-agent planning, we argue that it does not necessarily hold anymore in the context of learning. Indeed, we argue that agents which learn by reinforcement do not necessarily have to share the same payoff function. Remember, it is only through the reward function that the problem itself is defined, which means that it should include all influencing parameters such as progress of the task, but also eventual resource consumption and others. The problem is now that some of these factors may be global, while others, such as battery consumption, may only be known on a local scale. The property that some local information is hidden to the other agent is sometimes referred to as *imperfect monitoring*.

The reason why imperfect monitoring affects the definition of the reward function in multi-agent reinforcement learning as compared to multi-agent planning is the following : in planning scenarios, the reward itself is not distributed to the agents during execution, which means it does not give any additional information about the system. The total reward could actually be computed after the execution has terminated. In learning scenarios however, the reward signal is generated in each step and plays a major part in the algorithm itself. This difference is shown in Figure 8.4. Suppose an exploration scenario with two rovers on the planet Mars. The reward

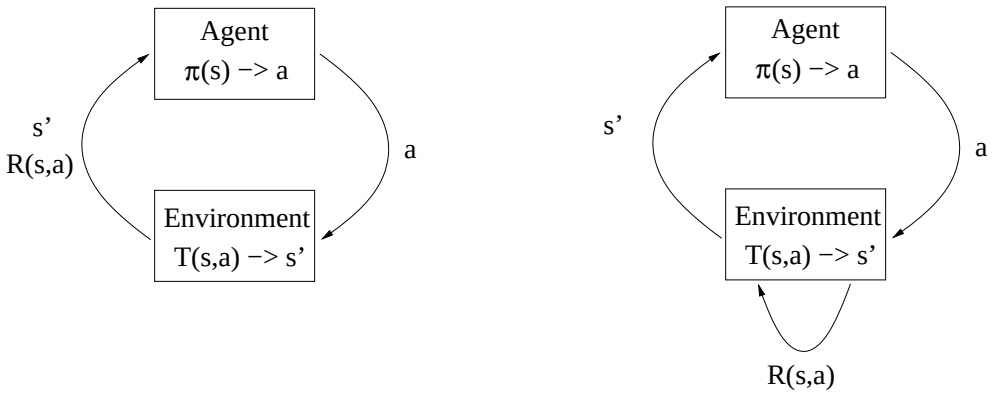

FIG. 8.4 – The fundamental difference between reinforcement learning (left) and planning (right). While agents are aware of their immediate reward in the first case, they have no knowledge about it during executing in the second case.

function integrates the progress of exploration, but also the global battery consumption. If there was just a single reward function, as it is commonly assumed for cooperative environments, calculating such a reward may already reflect an implicit coordination between agents. Sharing this reward with both agents gives each agent information about the other agents' battery status, thus distributing some global information that is otherwise not accessible. We argue that cooperative multi-agent reinforcement learning scenarios have to allow for individual reward functions, such as it is the case with the mutual notification algorithm.

8.5 Conclusion

Reinforcement learning for multi-agent system effectively means distributing the optimization process. This requires addressing a variety of issues, such as message passing between agents, coordination of actions, or synchronization of execution. Some important aspects, such as the issue of equilibrium selection in Nash Q-learning, have not been addressed so far, and we believe that there remains a lot of work to be done. Nevertheless, some of the concepts of multi-agent reinforcement learning have already been applied successfully to solve challenging problems, such as robotic soccer [RM01]. We pointed out that the definition of rewards in multi-agent reinforcement learning should be revised. We indeed believe that local reward functions have to be allowed in order to avoid implicit information sharing through the reward at runtime.

As opposed to the planning algorithms presented before, multi-agent reinforcement learning is much more related to the concepts of game theory, presented in Chapter 5, since there is no central entity anymore that could help coordinating the agents optimally. Again, we summarize the algorithms that we discussed in Figure 8.5.

	Mutual Notification, [SC04c]
Model	DEC-MDP
Assumptions	Communication, full observability, deterministic transitions, individual reward functions.
Horizon	Infinite
Solution	Optimal policies
Collaboration	Cooperative
Description	Q-learning with communication.

	Nash Q-learning, Correlated Q-learning, [HW03, GH03]
Model	DEC-MDP
Assumptions	Full observability, a single reward function.
Horizon	Infinite
Solution	Nash equilibrium
Collaboration	Competitive
Description	Q-learning and stochastic games.

	Bayesian Reinforcement Learning, [CB03]
Model	DEC-MDP
Assumptions	Full observability, a single reward function.
Horizon	Infinite
Solution	Nash equilibrium
Collaboration	Competitive
Description	Q-learning on belief states.

	Friend-or-foe Learning, [Lit01]
Model	DEC-MDP
Assumptions	Full observability, 2 players.
Horizon	Infinite
Solution	Optimal or min-max solution
Collaboration	Cooperative and competitive
Description	Q-learning or min-max learning.

	Gradient Ascent Reinforcement Learning, [PKMK00, BDC03]
Model	Distributed POMDPs
Assumptions	Factored state and action space
Horizon	Infinite
Solution	Locally optimal solution (equivalent to Nash equilibrium)
Collaboration	Cooperative
Description	Distributed gradient ascent Q-learning.

FIG. 8.5 – Solving DEC-POMDPs using reinforcement learning.

Chapitre 9

Experiments

9.1 Introduction

Although most of the algorithms we introduced in this thesis are provably optimal, it remains to be shown how they perform in comparison to other state-of-the-art techniques. This is the topic of the following chapter. We show in particular how the point-based dynamic programming and the heuristic search algorithms perform in comparison to previous dynamic programming algorithms on several test scenarios from the literature. We also show the performance of the multi-agent reinforcement learning algorithm on two simple cooperative robotic tasks. These results are by no means exhaustive. The performance of the algorithms for example heavily depends on their implementation, and the number of test scenarios currently in use among the multi-agent planning community is rather small. A valuable project would consists in summarizing and standardizing current test scenarios for DEC-POMDPs, similar to Anthony Cassandra's *"POMDP page"* [2].

9.2 Experiments Involving Planning

We first address the performance of the planning algorithms introduced in Chapter 6 and Chapter 7, namely the point-based dynamic programming approaches, and the heuristic search approaches. The experiments enable a comparison of the algorithms among each other, as well as with previous dynamic programming approaches.

9.2.1 Test Domains

We chose three common test domains for decentralized multi-agent problems from the literature in order to run our experiments. They have been defined in previous work by various authors, and are widely being used among the community.

9.2.1.1 Multi-agent Tiger Problem

The multi-agent tiger problem is an extension of the classical tiger problem for POMDPs (see Section 3.2.7.1). It has been introduced by [NTY+03] and involves 2 agents that listen at two doors. Behind one door lies a hungry tiger, and behind the other door are hidden untold riches. The agents however are untold the position of the tiger. Each agent may listen at its door, and thus increase its belief about the position of the tiger, or it may open a door. After a door has been

[2] `http ://pomdp.org/pomdp` - as of August 2006

opened by one of the agents, the system is reset to its initial state. In version A of the problem, a high reward is given if the agents jointly open the door with the riches, but a negative reward is given if any agent opens the door with the tiger. Listening incurs a small penalty. In version B, agents are not penalized in the special case where they jointly open the door of the tiger. The multi-agent tiger problem has 2 states, 3 actions and 2 observations. If the agents jointly

	Left	**Right**
⟨right, right⟩	+20	-50
⟨left, left⟩	-50	+20
⟨right, left⟩	-100	-100
⟨left, right⟩	-100	-100
⟨listen, listen⟩	-2	-2
⟨listen, right⟩	+9	-101
⟨right, listen⟩	+9	-101
⟨listen, left⟩	-101	+9
⟨left, listen⟩	-101	+9

FIG. 9.1 – Reward function A of the multi-agent tiger problem.

	Left	**Right**
⟨right, right⟩	+20	0
⟨left, left⟩	0	+20
⟨right, left⟩	-100	-100
⟨left, right⟩	-100	-100
⟨listen, listen⟩	-2	-2
⟨listen, right⟩	+9	-101
⟨right, listen⟩	+9	-101
⟨listen, left⟩	-101	+9
⟨left, listen⟩	-101	+9

FIG. 9.2 – Reward function B of the multi-agent tiger problem.

listen at the door, each agent receives the correct observation about the position of the tiger with probability 0.85, but with probability 0.15, an erroneous observation is produced. If one of the agents opens a door, observations have no value of information, since the systems is reset. The multi-agent tiger problem is a coordination problem. In order to maximize the expected reward, agents have to agree on which door to open. They may however have different beliefs about the position of the tiger, and this is precisely the fundamental difficulty when solving DEC-POMDPs.

9.2.1.2 Broadcast Channel Problem

The second setting simulates a simplified multi-access broadcast channel, where agents are situated at the two nodes of the channel. The problem has been introduced by [OW96], and formalized in the context of DEC-POMDPs by [HBZ04]. Each agent has to decide whether or not to send one of the messages from its message buffer. Sending is exclusive, which means that only one message can go through the channel at each time. If both agents try to send a message at the same time, a collision occurs, and the messages will remain in the buffer. The problem is partially observable and hence decentralized, since agents can only observe the state of their own message buffers, but do not know whether the other agent has also something to send or not. The common goal of the agents is to maximize the throughput of the system, with a reward of +1.0 given for any message that has been transmitted ; otherwise the reward is 0. In the experiments we conducted, there are 2 agents and 2 possible actions for each one of them (`send`, `not send`). The buffer size is 1, and the number of global states thus is 4, namely the cross-product of the local buffer states. There are 6 possible observations, characterized by the local buffer state (`empty`, `full`) and a status flag of the channel from the previous time step (`idle`, `active`, `collision`). New messages arrive with a rate of $p_1 = 0.9$ for agent 1, and $p_2 = 0.1$ for agent 2.

Buffer_t	Action	Buffer_{t+1}	Prob
`empty`	(any)	`empty`	$1.0 - p_i$
		`full`	p_i
`full`	`send`	`empty`	$1.0 - p_i$
		`full`	p_i
	`not send`	`empty`	0.0
		`full`	1.0

FIG. 9.3 – Channel problem : transition probabilities for each one of the buffers.

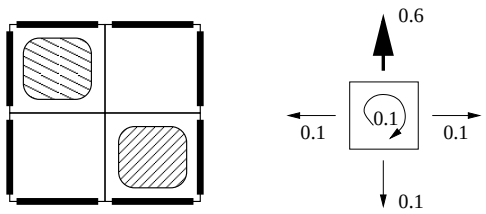

FIG. 9.4 – Meeting on a grid : 2 agents on a 2×2 grid, and the transition probabilities for action `North`.

9.2.1.3 Meeting on a Grid

In this problem, discussed by [BHZ05], two agents navigate on a two by two grid-world, without interfering with each other (see Figure 9.4). Their goal is to stay both on the same grid cell - which produces a reward of +1.0 - but their observation capabilities are limited and they do not see each other. The agents also have limited sensing capabilities concerning the environment : there are only 4 observations, indicating whether there is a wall to their left and/or to their right. Each agent has 5 actions, to move in any one of the four directions, or stay on its current cell, but transitions are stochastic, and the agent only moves with probability 0.6 in its intended direction. This is shown on the right part of Figure 9.4. The problem has 16 states, namely the cross product of the position of the two agents.

9.2.2 Results

The experiments we conducted enable different comparisons. First of all, the heuristic search algorithm MAA* is compared to the earlier dynamic programming approaches for finite-horizon problems. Then the point-based dynamic programming approach is evaluated against the heuristic search algorithm and Hansen et al.'s dynamic programming approach. Finally, the infinite-horizon approaches are compared to each other.

9.2.2.1 MAA* and Dynamic Programming

We tested the multi-agent heuristic search algorithm from Chapter 7 on two of the above problem settings, namely the multi-agent tiger problem and the broadcast channel problem, and we compared its performance with the current dynamic programming approach from [HBZ04]. We implemented both the MDP heuristic and the recursive approach on a 3.4GHz machine with 2GB of memory. Figure 9.5 and Figure 9.6 show results for both heuristics, namely the value of the optimal joint policy, the number of evaluated policy pairs, and the maximum memory requirements in terms of the size of the open list. MAA* with the MDP heuristic explores 33.556.500 policy pairs on the horizon-4 channel problem, which means 3% of the pairs that would have to be evaluated by a brute force approach. However, its actual memory requirements are much less important, because we can delete any suboptimal horizon-T policy pair after evaluating it. The maximum size of the open list thus never exeeds 1.038 policy pairs on that problem. Recursive MAA* performs better on all three test problems, although its heuristic is more complex. This is due to the additional pruning as a result of the tighter heuristic. Runtimes range from a few milliseconds (T=2) to several seconds (T=3) and up to several hours (T=4). We have

Problem	T=2	T=3	T=4
Tiger (A)	-4.00	5.19	4.80
	252	105.228	944.512.102
	8	248	19.752
Tiger (B)	20.00	30.00	40.00
	171	26.496	344.426.508
	8	168	26.488
Channel	2.00	2.99	3.89
	9	1.044	33.556.500
	3	10	1.038

FIG. 9.5 – MAA* using the MDP heuristic (shown are : value of optimal policy, number of evaluated policy pairs, max open list size)

Problem	T=2	T=3	T=4
Tiger (A)	-4.00	5.19	4.80
	252	105.066	879.601.444
	8	88	18.020
Tiger (B)	20.00	30.00	40.00
	171	26.415	344.400.183
	8	158	25.102
Channel	2.00	2.99	3.89
	9	263	16.778.260
	3	6	461

FIG. 9.6 – Recursive MAA* (shown are : value of optimal policy, number of evaluated policy pairs, max open list size)

also compared MAA* to the dynamic programming approach from [HBZ04], to our knowledge the only other non trivial algorithm that optimally solves DEC-POMDPs. Figure 9.7 shows the total number of policy pairs each algorithm has to evaluate, whereas Figure 9.8 compares the actual memory requirements. Although heuristic search does not necessarily consider less policy pairs on the sample problem, its memory requirements are much less important. One significant advantage of MAA* over the dynamic programming approach thus lies in the fact that it can tackle larger problems where dynamic programming will simply run out of memory. In addition,

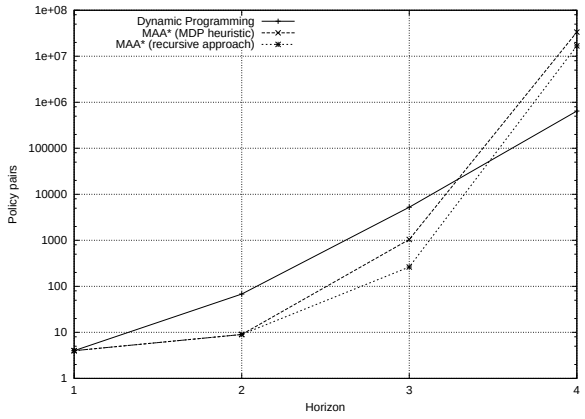

FIG. 9.7 – The number of evaluated policy pairs for dynamic programming, MAA* with the MDP heuristic, and recursive MAA* on the channel problem (logarithmic scale).

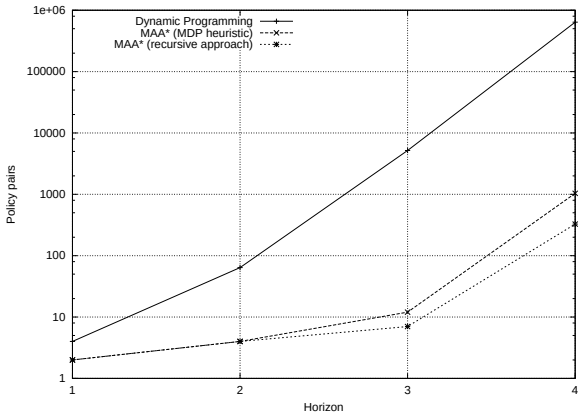

FIG. 9.8 – The actual memory requirements for dynamic programming, MAA* with the MDP heuristic, and recursive MAA* on the channel problem (logarithmic scale).

it avoids the computationally expensive linear programming part of the dynamic programming approach, necessary to identify dominated policy trees. Due to its anytime characteristic, MAA* is furthermore able to return an optimal solution after a very short time, although it might take much longer to guarantee its optimality. On the horizon-4 channel problem for example, the optimal solution is found after a few minutes already. An example solution is shown in Figure 9.9. We are equally able to compute suboptimal solutions for larger horizons, although the al-

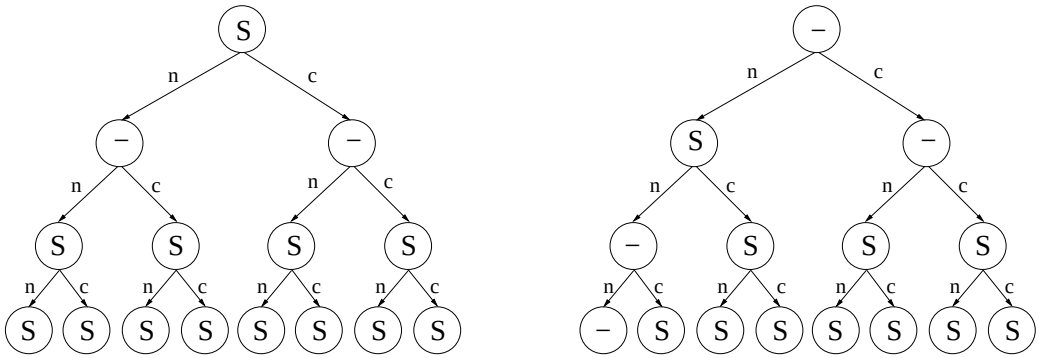

FIG. 9.9 – An optimal solution for the two-agent horizon 4 channel problem. The policy for agent 1 is shown on the left, the policy for agent 2 is shown on the right. The symbols are 'S' for sending a message, and '-' for not sending it. The subtree to the right is selected if a collision has been observed ('c'), otherwise the subtree to the left is chosen ('n').

gorithm will in general run out of time before terminating the computation. These results were first published in [SCZ05].

9.2.2.2 Point-based Dynamic Programming

We also tested the point-based approach from Chapter 6 against the previous heuristic search algorithm and Hansen et al.'s dynamic programming approach. We implemented both the exact version and an approximation, and we tested the algorithms on the multi-agent tiger problem as well as the broadcast channel problem. The approximate version samples a single prior joint policy (line 6 of Algorithm 30), but considers the exhaustive set of belief points for the association of policies to leaf nodes. The values were averaged over 10 trials.

We have seen that the major obstacle for the dynamic programming algorithm in [HBZ04] is memory. On the channel problem, the algorithm runs out of memory at horizon 5, and on the tiger problem, it does so at horizon 4. Figure 9.10 presents comparison results with both the exact and the approximate point-based multi-agent dynamic programming algorithm for the channel problem. Hansen *et al.*'s algorithm keeps 300 dominant policies at iteration 4 and cannot enumerate the exhaustive set of policies necessary for iteration 5. The exact point-based approach already keeps much fewer policies in memory, which means that most of the 300 policies identified by Hansen *et al.*'s approach lay in regions of the belief space that will never be visited. The approximate point-based approach is even less memory demanding. The number of policies kept by the point-based approaches also depends on the problem horizon, since a different horizon implies different prior policies and thus different belief sets. The exact approach was tested up to horizon 5, and the approximate approach up to horizon 8, which means that there are actually 5 (resp. 8) different curves for each case. Since the point-based algorithms have to consider much fewer policies, and because they do not include the linear programming part necessary to identify dominated policies, their runtime is usually expected to be shorter. Figure 9.11 shows a runtime comparison on the multi-access channel problem for all three approaches. The exact point-based dynamic programming approach is indeed faster than the previous dynamic programming approach, and the approximate point-based dynamic programming approach is even more competitive. The runtime of the point-based approximation surprisingly goes down

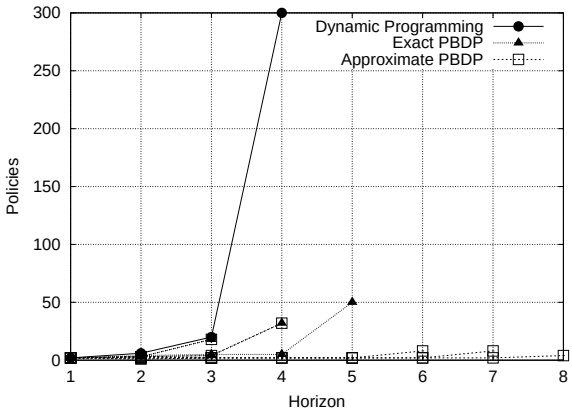

FIG. 9.10 – The number of useful policies evaluated by Hansen *et al.*'s dynamic programming approach, the exact point-based dynamic programming approach, and the approximate point-based dynamic programming approach on the multi-access channel problem. The size of the policy set for the point-based approaches depends on the horizon.

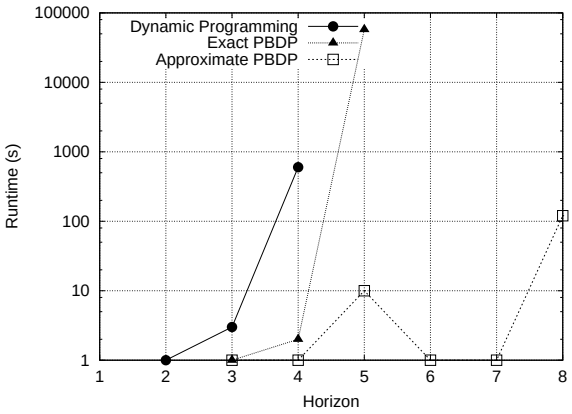

FIG. 9.11 – The effective runtime of the three approaches on the multi-access channel problem, measured in seconds and plotted on a logarithmic scale. The runtime of the approximate point-based dynamic programming approach is averaged over 10 trials.

for horizons 6 and 7. This might be explained by the fact that a different horizon results in a different prior policy, and thus in a different set of belief points. It is possible that a smaller set of policies is sufficient to cover up a larger set of belief points. Note that the ordinate is scaled logarithmically.

In Figure 9.12, we finally give a comparative overview of the solutions that can be obtained with the dynamic programming algorithm from [HBZ04], the heuristic search algorithm MAA* from [SCZ05], and the point-based approaches [SC06]. On the broadcast channel problem, the point-based dynamic programming approaches are definitely the most competitive algorithms, since they produce optimal or nearly optimal policies and enable to solve for larger problems. On the more difficult tiger problems, the point-based approaches are more competitive than the previous dynamic programming algorithm, but they do not permit to outperform the heuristic search algorithm.

9.2.2.3 Infinite-horizon MAA*

We tested the infinite-horizon heuristic search algorithm on two problems that have already been studied before, namely the broadcast channel problem, and the navigation task on a grid [BHZ05]. The discount rate for all problems is $\gamma = 0.9$. The highest possible discounted sum of rewards that can be attained for the channel problem assuming full global observability therefore is $\sum_t \gamma^t(1.0) = 10.0$, which would mean that a message could be transmitted at each time step. We can see in Figure 9.13 that a deterministic single-node FSC already achieves a value of about 9.0, although the problem is now decentralized and only partially observable. The bounded policy iteration approach for stochastic controllers produces less competitive policies. Figure 9.13 shows that the search algorithm is also more competitive on the navigation task than the policy iteration approach. However, it takes more time to converge and may in the end run out of memory. A

Channel	**T=3**	**T=4**	**T=5**	**T=6**	**T=7**	**T=8**
DP	2.99	3.89				
MAA*	2.99	3.89				
exact PBDP	2.99	3.89	4.79			
approximate PBDP	2.99	3.89	4.70	5.69	6.59	7.40

Tiger (A)	**T=2**	**T=3**	**T=4**			
DP	-4.00	5.19				
MAA*	-4.00	5.19	4.80			
exact PBDP	-4.00	5.19				
approximate PBDP	-4.00	5.19	4.80			

Tiger (B)	**T=2**	**T=3**	**T=4**			
DP	20.0	30.0				
MAA*	20.0	30.0	40.0			
exact PBDP	20.0	30.0				
approximate PBDP	20.0	30.0	40.0			

FIG. 9.12 – Values of the resulting policies for Hansen *et al.*'s dynamic programming approach, the MAA* approach, and the point-based multi-agent dynamic programming algorithms. The test scenarios were the multi-access broadcast problem, and two versions of the multi-agent tiger problem. Blanks indicate that the algorithms have either run out of memory or out of time.

solution for the navigation task is shown in Figure 9.14. The experimental results show that the advantage of using stochastic controllers, which have the theoretical ability to produce higher average rewards than deterministic ones [SJJ94], might be more than consumed by the local optimality of the algorithms that are required to compute them. In addition, the deterministic controllers with more than one node are degenerated versions of the one controller case. In the given example problems, having a larger memory does not necessary seem to help increasing the solution quality, which is why the value of the deterministic controllers do not increase with increasing size.

9.2.3 Discussion

We were able to demonstrate that the heuristic search algorithm may have significant advantages over the dynamic programming approach on finite-horizon problems. This is partly due to its ability to exploit a known start state distribution, and thus to exclude policy candidates that would only be useful in unreachable belief states. The same holds for the point-based dynamic programming approach, which enables the sampling of reachable belief points. The point-based approach also avoids the dynamic programming part, which is computationally expensive. The results of the infinite-horizon search approach indicate that, albeit stochastic controllers theoretically promise higher expected rewards, it might me much more difficult to actually optimize them.

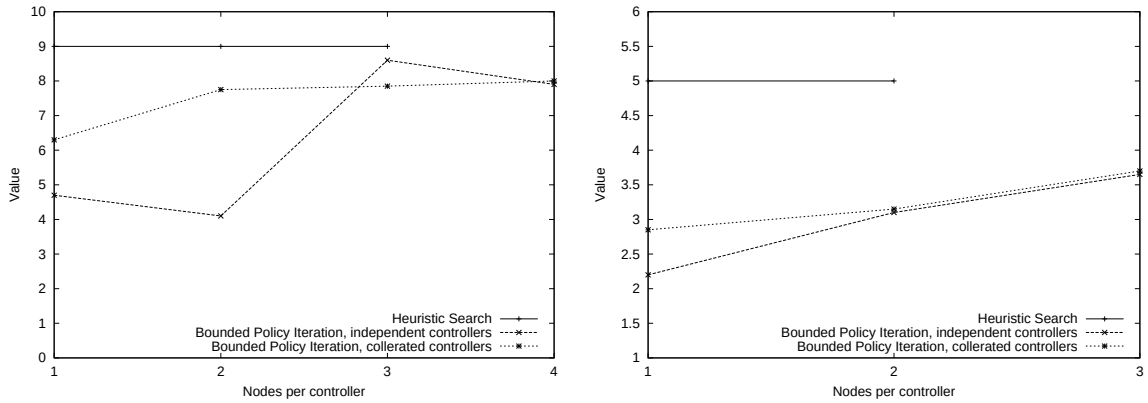

FIG. 9.13 – Value of optimal deterministic joint policy for the heuristic approach, and average value per trial run for two versions of bounded policy iteration on stochastic controllers. Left : Channel problem - Right : Meeting on a grid problem.

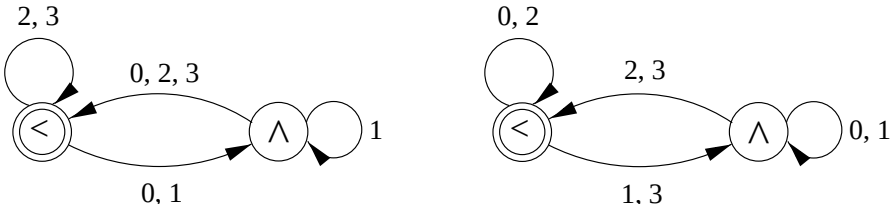

FIG. 9.14 – A 2-node solution for the navigation task where 2 agents have to meet on a grid. The symbol '$<$' represents moving to the left, and the symbol '$\wedge$' represents moving upwards. The observations are $0, 1, 2, 3$ indicating the presence or not of a wall at the left or the right of the agent, namely the four possible observations $(-; -), (wall; -), (-; wall), (wall; wall)$.

9.3 Experiments Involving Learning

Another set of experiments was conducted to evaluate the performance of the multi-agent reinforcement learning approach from Chapter 8.

9.3.1 Test Domains

The test domains for the learning approaches were a simplified soccer task, and a coordination problem commonly used in multi-agent learning scenarios.

9.3.1.1 Soccer Problem

The first learning task is a simplified soccer task : On a 5×5 grid, a team of two agents has to push a ball towards a specified goal tile, an action which produces a reward of $+10$ for each agent simultaneously (no distinction is made about which agent actually pushed the ball into the goal area). The available actions for each agent are `moveN`, `moveE`, `moveS`, `moveW`, and a `shoot` action, which moves the ball by two cells if it is placed next to the agent. The `shoot`-action was introduced to allow passing between agents. Each action has a small cost of -0.1. The exploration

strategy is an ϵ-greedy one with three different values for ϵ (0.3, 0.5, 0.7). The learning rate is $\gamma = 0.9$. The setting of the soccer problem is shown in the left side of Figure 9.15.

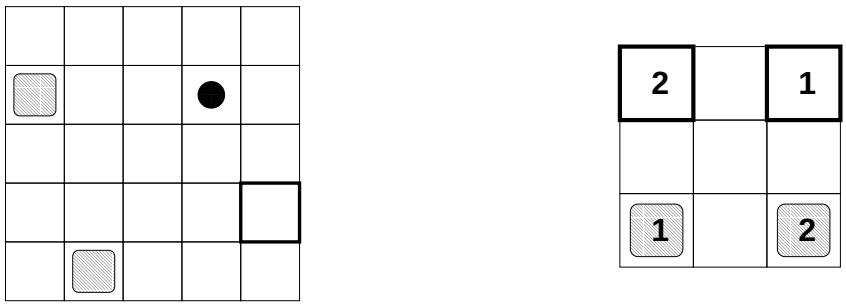

FIG. 9.15 – The soccer task with two agents and a ball, and the coordination task with two agents. The goal areas are the bold framed cells.

9.3.1.2 Coordination Problem

The second setting simulates a coordination task on a 3×3 grid, shown in the right side of Figure 9.15. Each agent has to reach an individual goal tile, which produces a reward of +10 for that agent. The available actions are `moveN`, `moveE`, `moveS`, `moveW`. No cell can be occupied by both agents at the same time, which makes coordination not trivial because the optimal paths for both agents intersect. The same coordination task has also been studied by [HW03] and others for non-cooperative agents. We used the same parameters as before, namely an ϵ-greedy exploration strategy with $\epsilon = 0.3, 0.5, 0.7$, and a learning rate of $\gamma = 0.9$.

9.3.2 Results

The first set of experiences was conducted on the soccer task. In Figure 9.16, we compare the cumulative number of communications for all three exploration rates with the identity function, which represents communication at every time step. In this experiment, the communication overhead increases slightly with higher exploration rates, but there are about eight to ten times less messages exchanged when compared to a communication at every learning step. The convergence of the joint policy value is shown in Figure 9.17. Again we compare the performance for all three exploration rates, and we also compare the algorithm to an uncoordinated Q-learning, which we know does not provably converge to an optimal solution. The experiments confirm the convergence of our algorithm and show the dependency of convergence speed and exploration.

A second set of experiments was conducted for the 2-robot coordination task on a grid. In Figure 9.18, the communication overhead is shown, again compared to the identity function. The experiments show that a higher exploration rate does not necessarily result in more communication : The algorithm with $\epsilon = 0.7$ is less communicative than with $\epsilon = 0.5$. This is especially interesting in comparison with the convergence of the joint policy value as shown in Figure 9.19 : the best convergence speed ($\epsilon = 0.7$) is obtained with only a medium communication overhead. Depending on the task, there might be an optimal tradeoff between convergence and communication. Finally, the experiments confirm the difficulty for an uncoordinated Q-learning to solve this task.

Figure 9.20 shows the number of communications for different thresholds : The no-threshold

case ($\delta = 0.0$) is compared to a fixed threshold ($\delta = 1.0, 2.0, 5.0$) as well as to a variable threshold (linearly decreasing threshold from 10.0 to 2.0, and exponentially decreasing threshold from 10.0 to 2.0). The convergence of the value function is shown in Figure 9.21. We can show that already the introduction of a light threshold ($\delta = 1.0$) decreases the communication overhead by a factor of 3 without affecting the convergence speed. Variable thresholds allow even further reductions of the number of communications, and compared to the worst case communication at each iteration, the number of communications can be reduced by a factor of 40 for the considered problem.

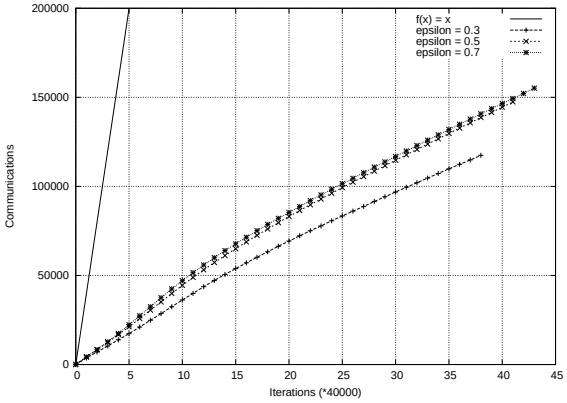

FIG. 9.16 – Soccer task : the total number of communications for three different exploration parameters ($\epsilon = 0.3$, $\epsilon = 0.5$, and $\epsilon = 0.7$) compared to the identity function $f(x) = x$ (communication at every time step).

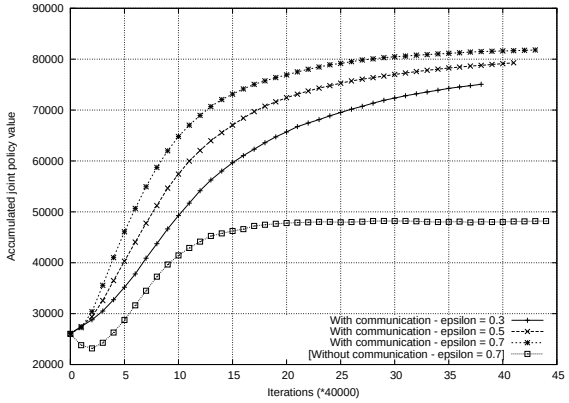

FIG. 9.17 – Soccer task : the total joint policy value summed over both agents and accumulated over all states, and for three different exploration parameters : $\epsilon = 0.3$, $\epsilon = 0.5$, and $\epsilon = 0.7$. The algorithm is also compared to a standard decentralized Q-learning without any explicit coordination or communication.

9.3.3 Discussion

Our experiments confirm first of all the convergence of the mutual notification algorithm, especially in comparison to a distributed Q-learning algorithm. They also show the influence of the exploration rate and the communication frequency on the speed of convergence. On the test examples we considered, agents were communicating about every third to to every fourth learning step, and the communication overhead could even further be reduced by avoiding communicating when only small changes in the value function were detected. It remains to be shown how the learning algorithm scales to larger problems.

9.4 Overall Discussion

The experimental results underline the optimality of our algorithms, and witness on the ability of current planning and learning algorithms to solve decentralized control problems. The major conclusion that has to be made is that as of today, the problems that can effectively be solved are very small in size. This can of course be explained by the worst-case complexity of the DEC-POMDP model itself. However, exploiting further structure, identifying more compact representations, and establishing smarter approximation techniques should help increasing the

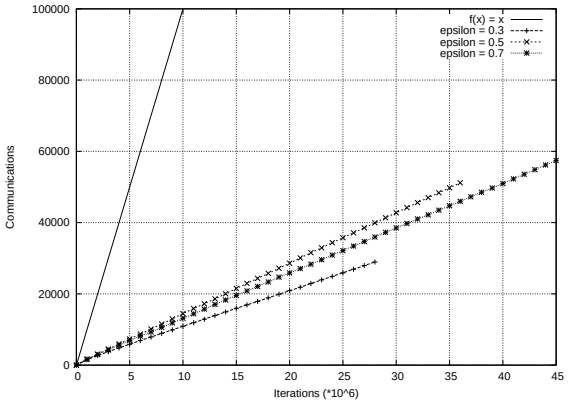

FIG. 9.18 – Coordination task : the total number of communications for three different exploration parameters ($\epsilon = 0.3$, $\epsilon = 0.5$, and $\epsilon = 0.7$), again compared to the identity function $f(x) = x$.

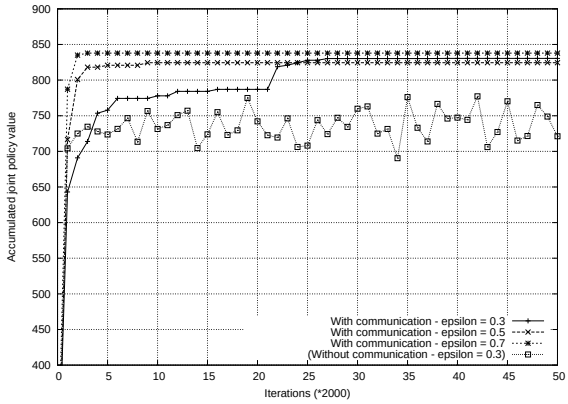

FIG. 9.19 – Coordination task : the total joint policy value summed over both agents and accumulated over all states, and for three different exploration parameters. The algorithm is again compared to a standard decentralized Q-learning without coordination.

size of problems that can effectively be solved.

Code was written in C++ and executed on a 3GHz machine with 1GB of RAM under Microsoft Windows XP. Solving the linear programs was done using ILOG CPLEX.

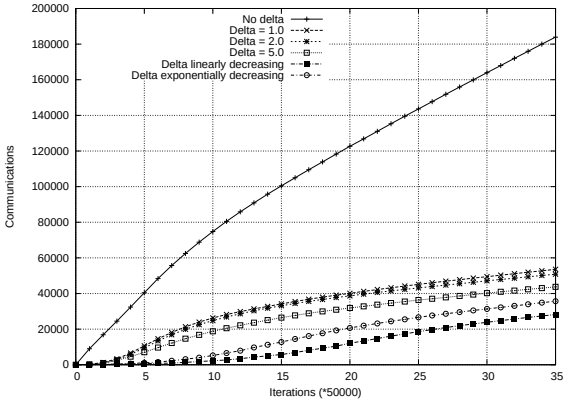

FIG. 9.20 – .
Soccer task : the total number of communications for three fixed thresholds, two variable thresholds, and the no-threshold case.

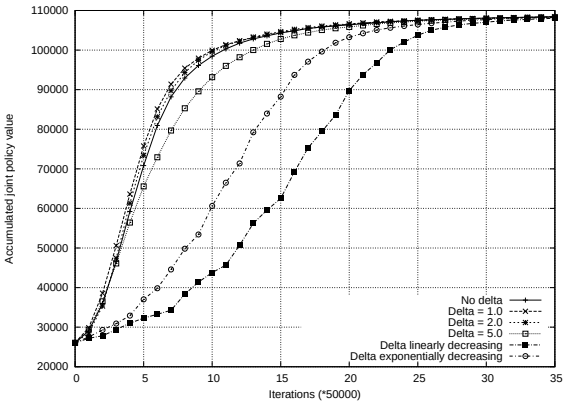

FIG. 9.21 – Soccer task : the joint policy value summed over both robots and all states for all 6 thresholds.

Chapitre 10

Conclusions

This thesis is an attempt to summarize and describe current theories for optimal decentralized decision making under uncertainty, while providing new theoretical insights and new algorithms to solve decentralized control problems. We focused on the DEC-POMDP model because of its expressiveness and its simplicity, and we identified four basic approaches to solve the DEC-POMDP, using game theory, dynamic programming, heuristic search, and reinforcement learning, although the distinction is not always sharp. We finally validated our algorithms experimentally, and compared their performance to algorithms from the literature.

10.1 Summary of Contributions

On the theoretical side, we identified two new problem classes, namely *symmetric* and *uniform* DEC-POMDPs. We discussed the issue of determining the *value of information* contained in a message exchange between agents, based on the DEC-POMDP's value function. We also defined the multi-agent version of the classical *credit assignment problem*. We furthermore emphasized the need for *individual reward functions* for multi-agent reinforcement learning.

On the algorithmic side, we introduced several novel techniques for solving DEC-POMDPs optimally. We made for the first time the link between *point-based dynamic programming* for POMDPs and dynamic programming for POSGs. We established for the first time a class of *heuristic search algorithms* for solving DEC-POMDPs. We also addressed the issue of *optimal reinforcement learning* for a special class of multi-agent problems.

Our intention was to establish a synthesis of current state-of-the-art research for solving decentralized control problems. This work is related to the research of several other people, and their work may provide some valuable additional information. Bernstein laid the theoretical ground for DEC-POMDP theory and its complexity in his PhD thesis [Ber05]. Nair addressed coordinating larger multi-agent teams in [Nai04]. Becker focused particularly on identifying and exploiting particular structure among multi-agent systems [Bec06]. As usual, there is a lot of place for future research, and we present some of the possible directions which we believe that are particular interesting in the next section.

10.2 Future Work

As the underlying DEC-POMDP theory for solving decentralized control problems is still very young, there remains a lot of work to be done in order to make it a valuable tool for real applications. Some of these issues are discussed below.

Extending Recent MDP and POMDP Techniques to DEC-POMDPs We believe that an important research effort should be made in order to adapt several recent techniques in MDP and POMDP theory to decentralized control theory. Bayesian networks for example can, and already have been [Gue03], extended to deal with multiple agents. The theory of semi-Markov decision processes can be adopted by the multi-agent community, where different types of agents can be expected to have different execution times of their actions. SMDPs can thus help coordinating different types of agents, without the need for growing the state space. An attempt of extending Dietterich's Taxi problem [Die00] to multiple agents, including SMDP theory, has recently been made by Ghavamzadeh and Mahadevan [GM02, GMM06]. Finally, it remains to be shown whether *multi-agent PSRs* can help solving decentralized control problems. The motivation here is the following : since multi-agent belief states - as opposed to single-agent ones - implicitly include a probabilistic look into the future (namely a distribution over the other agents' future policies), the notion of a *test* may have a nice equivalent in multi-agent domains.

Applying Smarter Pruning and Search Techniques to DEC-POMDPs Current dynamic programming and search techniques use a rather primitive version of policy elimination or heuristic search. It remains to be shown whether smarter techniques, such as incremental pruning for example, can help speeding up policy computation.

Merging DEC-POMDP Theory and Mono-agent Decision Making We mentioned earlier that the general decentralized POMDP is of doubly exponentially worst case complexity, and that it remains so far unfeasible for larger problems. We also showed that particular independence properties can help reducing this complexity barrier. However, lots of real world applications can be divided into parts of multi-agent and mono-agent decision making. Higher-level strategic planning for example could consist in solving computationally expensive DEC-POMDPs, while lower-level planning, such as path planning, could require only POMDP theory. An attempt should be made in combining these two theories, similar to the work in MDPs, where flexible decomposition techniques have been developed [Par98, MHK$^+$98].

Establishing Error Bounds for MAA* We already mentioned in a previous chapter that even approximating a DEC-POMDP given an error bound ϵ is still a very hard problem [RGR02]. This however is only an upper bound on the expected complexity. Several experiments have let to the conclusion that optimal solutions are often found very early in the search process, and that most part of the computation is then spend on guaranteeing that the solution is indeed the optimal one. If there was a way of estimating the error of the current joint policy with respect to all those policies that still have to be examined, then the search process could be dramatically improved. The heuristics we use in MAA* give an answer to the following question : given a policy for the first t steps, what is the highest reward we can expect in the last $(T-t)$ steps ? The heuristic function one would have to establish would have to give an answer to the following question : given a policy $\boldsymbol{q}^t$ of a certain horizon t, what is the highest loss of reward we can expect when we do not analyze all of its neighboring policies $\boldsymbol{q'}^t$ of the same horizon ? We believe

this to be a tricky question, since it requires comparing policies in policy space while estimating their distance in value function space.

Identifying the Topology of the Multi-agent Belief Space Both Hansen et al.'s previous and our point-based dynamic programming algorithms are based on identifying and pruning dominated policies in belief space. In order to enhance the linear programming part required for the first approach, as well as the search for realistic belief points required in the latter one, it might be helpful to have an idea of the repartition of the multi-agent belief states that can actually occur. This could for example help excluding entire regions of the belief space that are irrelevant to the decision making process. Some recent work on determining multi-agent belief states, such as [VNTY05], could help getting an insight into the issue of identifying relevant regions of the multi-agent belief space.

Generalizing Multi-agent Reinforcement Learning Distributing the optimization process is a major issue in multi-agent policy computation, particularly in distributed reinforcement learning. So far however, only limited attempts have been made, usually extending existing game theoretic and mono-agent techniques, to establish a generalized theory for multi-agent reinforcement learning. Single-agent reinforcement learning has extensively made use of single-agent planning techniques, such as Bellman's principle. If the same is true for multi-agent reinforcement learning, then some of the notions we developed earlier, such as determining multi-agent belief states, should be valuable when establishing for example a multi-agent belief-state MDP.

Bibliographie

[Alt00] Eitan Altman. Applications of markov decision processes in communication networks : A survey. Technical Report RR-3984, INRIA Sophia-Antipolis, 2000.

[Ast65] Karl J. Astrom. Optimal Control of Markov Decision Processes with Incomplete State Estimation. *Journal of Mathematical Analysis and Applications*, 10 :174–205, 1965.

[BBW01] Jonathan Baxter, Peter L. Bartlett, and Lex Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15 :351–381, 2001.

[BDC03] O. Buffet, A. Dutech, and F. Charpillet. Automatic generation of an agent's basic behaviors. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'03)*, 2003.

[BDG95] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1104–1111, 1995.

[Bec06] Raphen Becker. *Exploiting Structure in Decentralized Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2006.

[Bel57] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Ber05] Daniel S. Bernstein. *Complexity Analysis and Optimal Algorithms for Decentralized Decision Making*. PhD thesis, University of Massachusets Amherst, 2005.

[BGIZ02] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27(4) :819–840, 2002.

[BHZ05] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized pomdps. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[BM05] Aurélie Beynier and Abdel-Illah Mouaddib. A polynomial algorithm for decentralized markov decision processes with temporal constraints. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS'05)*, 2005.

[BM06] Aurélie Beynier and Abdel-Illah Mouaddib. An iterative algorithm for solving constrained decentralized markov decision processes. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, 2006.

[Bou99] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 478–485, 1999.

[BP96] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, 1996.

[Bro51] George W. Brown. Iterative solutions of games by fictitious play. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation*. Wiley, New York, 1951.

[BT01] Dimitri P. Bertsekas and John Tsitsiklis. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 2001.

[BT03] Ronen I. Brafman and Moshe Tennenholtz. Learning to coordinate efficiently : A model-based approach. *Journal of Artificial Intelligence Research*, 19 :11–23, 2003.

[BT04] Ronen I. Brafman and Moshe Tennenholtz. Efficient learning equilibrium. *Artificial Intelligence*, 159 :27–47, 2004.

[BT05] Dimitri P. Bertsekas and John Tsitsiklis. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 2005.

[Buf03] Olivier Buffet. *Une double approche modulaire de l'apprentissage par renforcement pour des agents intelligents adaptatifs*. PhD thesis, Université Henri Poincaré - Nancy I, 2003.

[BV02] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 2002.

[BZI00] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of markov decision processes. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, 2000.

[BZL04] Raphen Becker, Shlomo Zilberstein, and Victor Lesser. Decentralized markov decision processes with event-driven interactions. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'04)*, 2004.

[BZLG03] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Transition-independent decentralized markov decision processes. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'03)*, 2003.

[BZLG04] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Solving transition independent decentralized markov decision processes. *Journal of Artificial Intelligence Research*, 22 :423–455, 2004.

[CB98] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752, 1998.

[CB03] Georgios Chalkiadakis and Craig Boutilier. Coordination in multiagent reinforcement learning : A bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, 2003.

[Che99] Fangruo Chen. Decentralized supply chains subject to information delays. *Management Science*, 45 :1076–1090, 1999.

[CLZ97] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning : A simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI'13)*, 1997.

[Cou38] Antoine Augustin Cournot. *Recherches sur les Principes Mathematiques de la Theorie des Richesses*. Hachette, 1838.

[Cou02] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[CPS92] Richard W. Cottle, Jong-Shi Pang, and Richard E. Stone. *The Linear Complementarity Problem*. Academic Press, 1992.

[CSC02] Iadine Chadès, Bruno Scherrer, and François Charpillet. A heuristic approach for solving decentralized-POMDP : Assessment on the pursuit problem. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, 2002.

[DBC01] Alain Dutech, Olivier Buffet, and François Charpillet. Multi-agent systems by incremental gradient reinforcement learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, 2001.

[DFA99] Richard Dearden, Nir Friedman, and David Andre. Model based bayesian exploration. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, 1999.

[dFvR03] Daniela P. de Farias and Benjamin van Roy. The Linear Programming Approach to Approximate Dynamic Programming. *Operations Research*, 51(6) :850–865, 2003.

[Die00] Thomas G. Dietterich. An overview of MAXQ hierarchical reinforcement learning. *Lecture Notes in Computer Science*, 1864 :26–44, 2000.

[DP85] R. Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32 :505–536, 1985.

[Dra62] Alvin W. Drake. *Observation of a Markov Process Through a Noisy Channel*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering, 1962.

[DSW06] Thomas Degris, Olivier Sigaud, and Pierre-Henri Wuillemin. Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*, 2006.

[EMGST04] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the 3rd AAMAS*, 2004.

[ESK01] Russell C. Eberhart, Yuhui Shi, and James Kennedy. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[FH01] Zhengzhou Feng and Eric A. Hansen. Approximate planning for factored pomdps. In *Proceedings of the 6th European Conference on Planning (ECP'01)*, 2001.

[FZ04] Zhengzhou Feng and Shlomo Zilberstein. Region-based incremental pruning for pomdps. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI'04)*, 2004.

[GAZ04] Claudia V. Goldman, Martin Allen, and Shlomo Zilberstein. Decentralized language learning through acting. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, 2004.

[GB98] Héctor Geffner and Blai Bonet. Solving large POMDPs by real time dynamic programming. In *Working Notes Fall AAAI Symposium on POMDPs*, 1998.

[GD05] Piotr J. Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24 :49–79, 2005.

[GH03] Amy Greenwald and Keith Hall. Correlated-q learning. In *Proc. 20th International Conf. on Machine Learning*, 2003.

[GL05] AnYuan Guo and Victor Lesser. Planning for weakly-coupled partially observable stochastic games. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[GM02] Mohammad Ghavamzadeh and Sridhar Mahadevan. A multiagent reinforcement learning algorithm by dynamically merging markov decision processes. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, 2002.

[GM04] Mohammad Ghavamzadeh and Sridhar Mahadevan. Learning to communicate and act using hierarchical reinforcement learning. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, 2004.

[GMM06] Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. Hierarchical multiagent reinforcement learning. *Journal of Autonomous Agents and Multi-Agent Systems*, 13 :197–229, 2006.

[GMS03] Pierre Gérard, Jean-Arcady Meyer, and Olivier Sigaud. Combining latent learning and dynamic programming in macs. *European Journal of Operational Research*, 160 :614–637, 2003.

[Gue03] Carlos E. Guestrin. *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Stanford University, 2003.

[GZ03] Claudia V. Goldman and Shlomo Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, 2003.

[GZ04] Claudia V. Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems : Categorization and complexity analysis. *JAIR*, 22 :143–174, 2004.

[Han97] Eric A. Hansen. An Improved Policy Iteration Algorithm for Partially Observable MDPs. In *Proceedings of the 10th Conference on Neural Information Processing Systems*, 1997.

[Han98] Eric A. Hansen. Solving pomdps by searching in policy space. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, 1998.

[Har68] Garrett Hardin. The tragedy of the commons. *Science*, pages 1243–1248, 1968.

[Hau97a] Milos Hauskrecht. Dynamic decision making in stochastic partially observable medical domains : Ischemic heart disease example. In *Proceedings of AI in Medicine Europe (AIME'97)*, 1997.

[Hau97b] Milos Hauskrecht. *Planning and Control in Stochastic Domains with Imperfect Information*. PhD thesis, Department of Electrical Engineering and Computer Science - Massachusetts Institute of Technology, 1997.

[Hau00] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13 :33–94, 2000.

[HBZ96] Eric A. Hansen, Andrew G. Barto, and Shlomo Zilberstein. Reinforcement learning for mixed open-loop and closed-loop control. In *Proceedings of the 9th Conference on Neural Information Processing Systems (NIPS'96)*, 1996.

[HBZ04] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on Artificial Intelligence*, 2004.

[HMC03] Sergiu Hart and Andrieu Mas-Colell. Uncoupled dynamics do not lead to nash equilibrium. *American Economic Review*, pages 1830–1836, 2003.

[Ho80] Yu-Chi Ho. Team decision theory and information structures. *Proceedings of the IEEE*, 68(6) :644–654, 1980.

[How60] Ron Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[HW03] Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4 :1039–1069, 2003.

[HZ96] Eric A. Hansen and Shlomo Zilberstein. Anytime heuristic search. In *AAAI Fall Symposium on Flexible Computation in Intelligent Systems*, 1996.

[HZ01] Eric A. Hansen and Shlomo Zilberstein. LAO* : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129 :35–62, 2001.

[JJS94] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6), 1994.

[Jr.96] Arthur E. Bryson Jr. Optimal control - 1950 to 1985. *IEEE Control Systems*, 16(3) :26–33, 1996.

[Kae93] Leslie Pack Kaelbling. *Learning in Embedded Systems*. A Bradford Book, The MIT Press, 1993.

[KLC98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101 :99–134, 1998.

[KT01] Hiroaki Kitano and Satoshi Tadoroko. Robocup-rescue : A grand challenge for multiagent and intelligent systems. *AI Magazine*, 22(1) :39–51, 2001.

[LCK95] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments : Scaling up. In *Proceedings of the 12th International Conference on Machine Learning*, 1995.

[Lit94] Michael Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ICML'94)*, 1994.

[Lit01] Michael Littman. Friend-or-foe q-learning in general-sum games. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*, 2001.

[LLB06] Asher Lipson and Kevin Leyton-Brown. Empirically evaluating multiagent reinforcement learning algorithms. *Machine Learning*, 2006.

[Lov91] William S. Lovejoy. Computationally feasible bounds for partially observed markov decision processes. *Operations Research*, 39(1) :162–175, 1991.

[LR00] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conf. on Machine Learning*, pages 535–542. Morgan Kaufmann, San Francisco, CA, 2000.

[LSS01] Michael Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Proceedings of the 14th International Conference on Advances in Neural Information Processing Systems (NIPS'01)*, 2001.

[MB01] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*, 2001.

[MC68] D. Michie and R. A. Chambers. Boxes : An experiment in adaptive control. *Machine Intelligence*, 2 :137–152, 1968.

[MHC03] Omin Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2) :5–34, 2003.

[MHK+98] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled markov decision processes. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, 1998.

[Min54] Marvin L. Minsky. *Theory of Neural-Analog Reinforcement Systems and its Application to the Brain-Model Problem*. PhD thesis, Princeton University, 1954.

[Min61] Marvin L. Minsky. Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49 :8–30, 1961.

[MKKC99] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony Cassandra. Solving POMDPs by Searching the Space of Finite Policies. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, 1999.

[Mon82] George E. Monahan. A survey of partially observable markov decision processes : Theory, models, and algorithms. *Management Science*, 28(1) :1–16, 1982.

[MP43] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5 :115–137, 1943.

[MP69] Marvin L. Minsky and Seymour Papert. *Perceptrons : An Introduction to Computational Geometry*. MIT Press, 1969.

[MS01] Maja J. Mataric and Gaurav S. Sukhatme. Task-allocation and coordination of multiple robots for planetary exploration. In *Proceedings of the 10th International Conference on Advanced Robotics*, 2001.

[Mye97] Roger B. Myerson. *Game Theory - Analysis of Conflict*. Harvard University Press, 1997.

[Nai04] Ranjit Nair. *Coordinating Multiagent Teams in Uncertain Domains using Distributed POMDPs*. PhD thesis, University of Southern California, 2004.

[Nas51] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2) :286–295, 1951.

[Nil80] Nils J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Co., Palo Alto, 1980.

[NTY+03] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized pomdps : Towards efficient policy computation for multiagent settings. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

[NVTY05] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed pomdps : A synthesis of distributed constraint optimization and pomdps. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, 2005.

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[OW96] James M. Ooi and Gregory W. Wornell. Decentralized control of a multiple access broadcast channel : Performance bounds. In *Proceedings of the 35th Conference on Decision and Control*, 1996.

[Par98] Ronald Parr. Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, 1998.

[PB03] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Proceedings of the 17th Conference on Neural Information Processing Systems (NIPS'03)*, 2003.

[Pea90] Judea Pearl. *Heuristique - Strategies de recherche intelligente pour la résolution de problèmes par ordinateur*. Cepadues-Editions, 1990.

[PGT03] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration : An anytime algorithm for pomdps. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003.

[PKMK00] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Kaelbling. Learning to cooperate via policy search. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, 2000.

[Poh73] I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5 :127–140, 1973.

[PS02] Leonid Peshkin and Virginia Savova. Reinforcement learning for adaptive routing. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2002.

[PT82] Christos H. Papadimitriou and John N. Tsitsiklis. On the complexity of designing distributed protocols. *Information and Control*, 53 :211–218, 1982.

[PT86] Christos H. Papadimitriou and John N. Tsitsiklis. Intractable problems in control theory. *SIAM Journal on Control and Optimization*, 24(4) :639–654, 1986.

[PT87] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processses. *Mathematics of Operations Research*, 12(3) :441–450, 1987.

[PT02] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem : Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, pages 389–423, 2002.

[Put94] Martin L. Puterman. *Markov Decision Processes - Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[RGR02] Zinovi Rabinovich, Claudia V. Goldman, and Jeffrey S. Rosenschein. Non-approximability of decentralized control. Technical Report CS2002-29, Hebrew University Jerusalem, 2002.

[RM01] Martin Riedmiller and Artur Merke. Karlsruhe brainstormers - a reinforcement learning approach to robotic soccer. In *RoboCup-2001 : Robot Soccer World Cup V, LNCS*. Springer, 2001.

[RMS02] Martin Riedmiller, Andrew Moore, and Jeff Schneider. Reinforcement learning for cooperating and communicating reactive agents in electrical power grids. In *Balancing Reactivity and Social Deliberation in Multi-agent Systems, Lecture Notes in Artificial Intelligence*. Springer, 2002.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence : A Modern Approach*. Prentice Hall, 1995.

[Ros58] Frank Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6) :386–408, 1958.

[Sam59] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, pages 210–229, 1959.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, Cambridge, MA, 1998.

[SBL06] Jiaying Shen, Raphen Becker, and Victor Lesser. Agent interaction in distributed pomdps and its implications on complexity. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*, 2006.

[SC02] Bruno Scherrer and François Charpillet. Cooperative co-learning : A model-based approach for solving multi agent reinforcement problems. In *Proceedings of the 14th International Conference on Tools with Artificial Intelligence (ICTAI'02)*, 2002.

[SC04a] Daniel Szer and François Charpillet. Communication et apprentissage par renforcement pour une équipe d'agents. In *Proceedings des Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2004)*, 2004.

[SC04b] Daniel Szer and François Charpillet. Coordination through mutual notification in cooperative multiagent reinforcement learning. Technical Report A04-R-051, LORIA - INRIA, Nancy, 2004.

[SC04c] Daniel Szer and François Charpillet. Improving coordination with communication in multiagent reinforcement learning. In *Proceedings of the 16th International Conference on Tools with Artificial Intelligence (ICTAI'04)*, 2004.

[SC05] Daniel Szer and François Charpillet. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*, 2005.

[SC06] Daniel Szer and François Charpillet. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, 2006.

[SCZ05] Daniel Szer, François Charpillet, and Shlomo Zilberstein. MAA* : A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, 2005.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27 :379–423 and 623–656, 1948.

[Sha50] Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41 :256–275, 1950.

[Sig04] Olivier Sigaud. Comportements adaptatifs pour des agents dans des environnements informatiques complexes. *Habilitation à Diriger des Recherches de l'Université Paris 6*, 2004.

[SJJ94] Satinder Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *Proceedings of the 11th International Conference on Machine Learning (ICML'94)*, 1994.

[SJR04] Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations : A new theory for modeling dynamical systems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI'04)*, 2004.

[SL99] Csaba Szepesvári and Michael Littman. A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation*, 11(8) :2017–2059, 1999.

[SLJ+03] Satinder Singh, Michael Littman, Nicholas Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, 2003.

[Son71] Edward J. Sondik. *The optimal control of partially observable Markov decision processes*. PhD thesis, Stanford University, 1971.

[SOZ+05] Yasuhiro Shirai, Andrew J. Osgood, Yuming Zhao, Kevin F. Kelly, and James M. Tour. Directional control in thermally driven single-molecule nanocars. *Nano Letters*, 5(11), 2005.

[SPG03] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning : A critical survey. Technical report, Stanford University, 2003.

[SPS99] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs : A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2) :181–211, 1999.

[SS73] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21 :1071–1088, 1973.

[SS05] Trey Smith and Reid Simmons. Point-based pomdp algorithms : Improved analysis and implementation. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, 2005.

[SV05] Matthijs T. J. Spaan and Nikos Vlassis. Perseus : Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24 :195–220, 2005.

[SZ05] Sven Seuken and Shlomo Zilberstein. Models and algorithms for decentralized control of multiple agents. Technical Report 2005-068, University of Massachusetts, 2005.

[Tan97] Ming Tan. Multi-agent reinforcement learning : Independent vs. cooperative learning. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 487–494. Morgan Kaufmann, 1997.

[Tho11] Edward L. Thorndike. *Animal Intelligence*. Hafner, Darien, 1911.

[Tho05] Vincent Thomas. *Proposition d'un formalisme pour la construction automatique d'interactions dans les systèmes multi-agent réactifs*. PhD thesis, Université Henri Poincaré - Nancy I, 2005.

[Tur59] Alan Turing. Computing machinery and intelligence. *Mind*, 59 :544–546, 1959.

[TW00] Kagan Tumer and David H. Wolpert. Collective intelligence and braess' paradox. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI'00)*, 2000.

[VNLP02] Katja Verbeeck, Ann Nowé, Tom Lenaerts, and Johan Parent. Learning to reach the pareto optimal nash equilibrium as a team. In *Proceedings of the 15th Australian Joint Conference on Artificial Intelligence, Canberra, Australia*, volume 2557 of *Lecture Notes in Computer Science*, pages 407–418. Springer-Verlag Heidelberg, 2002.

[vNM44] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[VNTY05] Pradeep Varakantham, Ranjit Nair, Milind Tambe, and Makoto Yokoo. Winning back the cup for distributed pomdps : Planning over continuous belief spaces. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

[Was96] Richard Washington. Incremental markov-model planning. In *Proceedings of the 8th International Conference on Tools with Artificial Intelligence*, 1996.

[Wat89] C. J. C. H. Watkins. *Learning from delayed rewards.* PhD thesis, King's College of Cambridge, UK., 1989.

[WD92] C. J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8 :279–292, 1992.

[Wei65] Joseph Weizenbaum. Eliza - a computer program for the study of natural language communication between man and machine. *Communications of the Association for Computing Machinery*, 9(1) :36–45, 1965.

[Wei99] Gerhard Weiss, editor. *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[Woo01] Michael Wooldridge. *Introduction to MultiAgent Systems*. John Wiley and Sons, 2001.

[WWT99] David H. Wolpert, Kevin R. Wheeler, and Kagan Tumer. General principles of learning-based multi-agent systems. In *Proceedings of the 3rd International Conference on Autonomous Agents*, 1999.

[ZWBM02] Shlomo Zilberstein, Richard Washington, Daniel S. Bernstein, and Abdel-Illah Mouaddib. Decision-theoretic control of planetary rovers. In M. Beetz et al., editor, *Plan-Based control of Robotic Agents*, number 2466 in LNAI, pages 270–289. Springer, 2002.