A

Annexes : CV détaillé

Nom patronymique : DUTECH **Prénoms** : Alain, Edgé

Date et lieu de naissance : 16 mars 1970 à EPINAL (88)
Nationalité : Français **Situation de famille** : Célibataire

Adresse Personnelle : 34 rue Sadi Carnot, 54220 Malzéville
Tél : 03.83.21.42.32
email : Alain.Dutech@loria.fr

Situation actuelle : **Chargé de Recherche, 1ère classe, INRIA**

A.1 Fonctions précédentes

- depuis sept. 01 : chargé de recherche INRIA
- sept. 00 à sept. 01 : ATER à l'ESIAL, Université H. Poincaré, Nancy I
- sept. 99 à sept. 00 : ATER à l'ESSTIN, Université H. Poincaré, Nancy I
- avr. 99 à sept. 99 : Post-Doc au LORIA, Nancy (bourse INRIA)

A.2 Diplômes - Titres Universitaires

Doctorat d'Intelligence Artificielle Mention : Très Honorable

Titre : "**Apprentissage d'environnements : Approches Cognitives et Comportementales**"
Directeur de Thèse : **Manuel Samuelides (ENSAE)**

Date : **1er février 1999** Lieu : **Toulouse**
Jury : C. Barouille, ONERA/CERT Toulouse (président)
P. Gaussier, ETIS/ENSEA Cergy (rapporteur)
J.-P. Haton, Université H. Poincaré Nancy (rapporteur)
A. Lanusse, Ecole Polytechnique Palaiseau
M. Samuelides, ENSAE Toulouse

DEA "Informatique Fondamentale et Parallélisme" Mention : Bien

Date : **1993** Lieu : **Toulouse**
Stage effectué au GTRI d'Atlanta (USA) sous la direction de John Gilmore sur l'apprentissage automatique de connaissances dans les systèmes à base de règles.

Ingénieur : Ecole Nationale Supérieure de l'Aéronautique et de l'Espace

Spécialisation : **Informatique**
Date : **1992** Lieu : **Toulouse**
Stage de fin d'étude effectué à l'Université de Sydney (Australie) sous la direction de Marwan Jabri sur la simulation et l'utilisation de réseaux de neurones artificiels pour la reconnaissance de la parole.

A.3 Publications

Revues internationales de rang A (selon classification ANU)

1. An investigation into Mathematical Programming for Finite Horizon Decentralized POMDPs. R. Aras and A. Dutech. *Journal of Artificial Intelligence Research*, Vol 37, pp 329-396, 2010. – *rang A*.
2. Shaping multi-agent systems with gradient reinforcement learning. Buffet O., Dutech A. et Charpillet F. *Journal of Autonomous agents and Multi-agent Systems*, Vol 15 :2, pp 197-220, 2007 – *rang A*.

Revues nationales

1. Etude de différentes combinaisons de comportements adaptatives. Buffet O., Dutech A. et Charpillet F. *Revue d'Intelligence Artificielle (RIA)*, volume 20(2-3), pp 311–344, 2006.
2. Développement autonome des comportements de base d'un agent. Buffet O., Dutech A., Charpillet F. *Revue d'Intelligence Artificielle (RIA)*, volume 19(4-5), pp 603–632, 2005.
3. Apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés. Dutech A. et Samuelides. M. *Revue d'Intelligence Artificielle, RIA*, volume 17(4), pp 559–589, 2003.

Chapitres de livre

1. Partially Observable Markov Decision Processes. Dutech A. et Scherrer B. Chapitre 7 de "*Markov Decision Processes in Artificial Intelligence*", Edité par O. Sigaud et O. Buffet, *ISTE Ld.*, *Wiley & Sons Inc.*, 2010.
2. Processus Décisionnels de Markov Partiellement Observables. Dutech A. et Scherrer B. Chapitre 3 de "*Processus décisionnels de Markov en intelligence artificielle*", vol 1, édité par O. Sigaud and O. Buffet, Hermes Science Publishing, 2008.

Conférences de rang A et A+ (selon classification ANU)

1. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. R. Aras, A. Dutech et F. Charpillet. *Int. Conf. on Automated Planning & Scheduling (ICAPS'07)*, Rhode Island, 2007 – *rang A+, 31,2%.*
2. Cooperation in stochastic games through communication. R. Aras, A. Dutech et F. Charpillet. *In Proc. of the fourth Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'05)*, Utrecht, Nederland, 2005 – *rang A+, 24,5%.*
3. Self-Growth of basic behaviors in an action selection based agent.. O. Buffet, A. Dutech et F. Charpillet. *In Proc. of Int. Conf. on Simulation of Adaptive Behavior (SAB'04)*, Los Angeles, 2004 – *rang A, 25%.*
4. Automatic Generation of an Agent's Basic Behaviors. Buffet O., Dutech A., Charpillet F. *In Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, Melbourne, 2003 – *rang A+, 24,7%.*
5. Adaptive Combination of Behaviors in an Agent. Buffet O., Dutech A., Charpillet F. *European Conference on Artificial Intelligence, (ECAI'02)*, Lyon, 2002 – *rang A, 27,2%.*
6. Learning to weigh basic behaviors in Scalable Agents. Buffet O., Dutech A., Charpillet F. *Autonomous Agents and Multi-Agent System, (AAMAS'02)*, Bologna, Italy, 2002 – *rang A+, 26,8%.*
7. Multi-Agent systems by incremental gradient reinforcement learning. Alain Dutech, Olivier Buffet and François Charpillet. *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, 2001 – *rang A+, 25%.*
8. Incremental reinforcement learning for designing multi-agent systems. Olivier Buffet, Alain Dutech and François Charpillet. *Proceedings of the 5th International Conference on Autonomous Agents (poster session), (AAMAS'01)*, Montréal, may 2001 – *rang A+.*
9. Solving POMDPs using selected past events. Alain Dutech. *European Conference on Artificial Intelligence, (ECAI'00)*, Berlin, 2000 – *rang A, 30%.*

Autres conférences avec comité de lecture et actes

1. Modèles stochastiques de la prise de décision collective. A. Dutech. *Colloque annuel de l'Association pour la Recherche Cognitive (ARCo'07)*, Nancy, 2007.

2. Une méthode de programmation linéaire mixte pour les POMDP décentralisés à horizon fini. R. Aras, A. Dutech et F. Charpillet. *Journées Françaises de Planification, Décision, Apprentissage (JFPDA'07)*, Grenoble, 2007.
3. Apprentissage par Renforcement et Théorie des Jeux pour la coordination de Systèmes Multi-Agents. A. Dutech, R. Aras et F. Charpillet. *Dans Colloque Africain sur la Recherche en Informatique - CARI 2006*, Cotonou, Bénin, 2006.
4. Efficient Learning in Games. R. Aras, A. Dutech and F. Charpillet. *(Poster) In Actes de la huitième Conf. Francophone sur l'Apprentissage (CAp'06)*, Trégastel, France, 2006.
5. Cooperation through communication in decentralized Markov games. R. Aras, A. Dutech et F. Charpillet. *In International Conference on Advances in Intelligent Systems - Theory and Applications - (AISTA'2004)*, Kirchberg, Luxembourg, 2004.
6. Stigmergy in Multi Agent Reinforcement Learning. R. Aras, A. Dutech et F. Charpillet. *In Fourth International Conference on Hybrid Intelligent Systems (HIS'2004)*, Kitakyushu, Japan, 2004.
7. On the use of Artificial Intelligence for Prognosis and Diagnosis in the PROTEUS E-maintenance platform. PROTEUS WP2 Team (L. Déchamp, A. Dutech, T. Montroig, X. Qian, D. Racoceanu, I. Rasovska, B. Brézillon, F.Charpillet, J-Y. Jaffray, N. Moine, B. Morello, S. Müller, G. Nguengang, N. Palluat, L.Pelissier). *In Proc. of Int. Conf. on Mechatronics and Robotics, (MECHROB'04)*, Aachen, Deutchland, 2004.
8. Proteus, des web services pour les systemes de maintenance. Rebeuf, Xavier and Blanc, Nicolas and Charpillet, François and Chevé, Denis and Dutech, Alain and Lang, Christophe and Pélissier, Loïc and Thomesse, Jean-Pierre. *In Actes des Nouvelles Technologies de la Répartition - NOTERE'04*, Saidia, Maroc, 2004.
9. Développement autonome des comportements de base d'un agent. A. Dutech, O. Buffet et F. Charpillet. *In Actes de la Conférence d'Apprentissage (CAp'04)*, Montpellier, 2004.
10. Apprentisage par renforcement pour la conception de systèmes multi-agents réactifs. A. Dutech, O. Buffet et F. Charpillet. *Journées Francophones des Systèmes Multi-Agents - Hors série Revue RSTI*, Hammamet, 2003.
11. Self-organizing autonomous robot controler. Alain Dutech. *Proceedings of the EIS'98 International Symposium on Engineering of Intelligent Systems*, Tenerife, Spain, février 1998.
12. World Modeling by the Fusion of Simpler Models. Alain Dutech and Manuel Samuelides. *Proceedings of the ICS'96 International Conference on Artificial Intelligence*, Kaohsiung, Taiwan, décembre 1996.
13. Detection of word-boundaries from continuous phoneme streams using simple recurrent neural networks. Laurens Leerink, Marwan Jabri and Alain Dutech. *Proceedings of the fourth australian international conference on speech science and technology*, pp 162-166, december 1992, Australia.

Workshops avec comité de lecture, avec ou sans actes

1. Quadratic programming for Multi-Target Tracking. R. Aras, A. Dutech et F. Charpillet. *MSDN Workshop at AAMAS'09*, Budapest, 2009.
2. Apprentissage par renforcement et jeux stochastiques à information incomplète. R. Aras et A. Dutech. *Journées PDMIA*, Lille, 2005.

3. A Self-Made Agent Based on Action-Selection. Buffet O., Dutech A. *In Proceedings of 6th European Workshop on Reinforcement Learning, EWRL-6*, Nancy, 2003.
4. Learning to use contextual information for solving partially observable Markov decision problems. Dutech A. and Scherrer B. *5th European Workshop on Reinforcement Learning, EWRL-5*, Utrecht, 2001.
5. Looking for scalable agents. Buffet O. and Dutech A. *5th European Workshop on Reinforcement Learning, EWRL-5,* Utrecht, 2001.
6. Learning dynamical extensions of observation state in a partially observed environment. Alain Dutech et Manuel Samuelides. *Workshop on Learning*, Snowbird, USA, 1999.

Communications diverses

1. IA et Cognition Incarnée. A. Dutech. *Conférence au Lycée Jeanne d'Arc*, Remiremont, mars 2010.
2. Tutorial sur les POMDP. A. Dutech. *Séminaires de l'équipe TAO*, Orsay, janvier 2010.
3. Tutorial sur les POMDP. A. Dutech. *Atelier sur les MDP, RFIA'2010*, Caen, janvier 2010.
4. Participation table ronde "Science en tête, Cerveau en fête" , Saint-Dizier, Samedi 22 Novembre 2008.
5. Intelligence Artificielle : espèce en voie d'apparition ? A. Dutech et N. Fatès. *Conférence dans le cadre de la fête de la science à Saint-Dizier*, Saint-Dizier, octobre 2007.
6. Self-Growing of Basic Behaviors by an Agent using Reinforcement Learning. Olivier Buffet, François Charpillet et Alain Dutech. *MAIA-UMASS Workshop*, Nancy, juin 2004.
7. Résoudre les POMDP en utilisant des informations contextuelles. Alain Dutech. *Journées "Stochastiques", Université Laval*, Québec, Canada, 3 et 4 mars 2003.
8. Modèles Stochastiques pour la Décision. Alain Dutech. *Journées du Réseau CogniEST, Traitement numériques issus de la Biologie*, Metz, 23 et 24 mars 2002.
9. Coopération par Apprentissage dans les systèmes artificiels. Alain Dutech. *Assemblée Générale du Réseau CogniEST*, La petite Pierre, 18 octobre 2001.
10. Reinforcement Learning for the design of multi-agent systems. Alain Dutech, Olivier Buffet. *Séminaire LORIA-NASA, Robotique Mobile sur Mars*, novembre 2000.
11. Apprendre à s'adapter dans les modèles de Markov. Alain Dutech. *Actes de la Journée sur l'Adaptation*, INRA Champenoux, mai 2000.

Thèses, Rapports techniques

- An investigation into Mathematical Programming for Finite Horizon Decentralized POMDPs. Raghav Aras, et Alain Dutech. *Rapp technique de l'INRIA, No RR-7066*, 2009.
- Using linear programming duality for solving finite horizon Dec-POMDPs. Raghav Aras, Alain Dutech, et François Charpillet. *Rapp technique de l'INRIA, No RR-6641*, 2008.
- Apprentissage d'environnement : Approches cognitives et comportementales. Alain Dutech. *Thèse de l'ENSAE.* Toulouse, 1999.

A.4 Encadrement

Thèse

- ▷ **Thèse d'Elham Ghassemi, Co-encadrement avec Y. Boniface et B. Girau**
  sujet : Mécanismes fonctionnels du Cervelet dans la commande motrice
  Thèse commencée en septembre 2008.
  Cette thèse s'intéresse aux fonctions du cervelet lors de la commande motrice occulaire et en particulier sur la correction et l'adaptation de la commande issue du colliculus supérieur.
- ▷ **Thèse de Raghav Aras, Co-encadrement avec F. Charpillet**
  sujet : Communications dans les jeux stochastiques décentralisés
  Septembre 2003 - Soutenue le 23 octobre 2008.
  Cette thèse s'intéresse à l'utilisation de la programmation mathématique dans le cadre des Processus Décisionnels de Markov Partiellement Observables Décentralisés (Dec-POMDP). En s'inspirant de la théorie des Jeux et de considérations combinatoires, Raghav à proposés plusieurs algorithmes originaux de résolution exacte des Dec-POMDP et a apporté des connaissances importantes sur la structure de ces solutions.
- ▷ **Thèse d'Olivier Buffet, Co-encadrement avec F. Charpillet**
  sujet : Apprentissage comportemental pour la conception de systèmes multi-agents
  Septembre 2001 - Soutenue le 10 septembre 2003.
  Cette thèse porte sur l'Apprentissage dans les Systèmes Multi-Agents. En utilisant le formalisme des processus décisionnels de Markov, nous travaillons sur des algorithmes de conception automatique de systèmes multi-agents par un processus incrémental et hiérarchique.

Stage de DEA

- ▷ **M2R ENSI de Tunis, de Selma Belgacem. Co-encadrement avec Y. Boniface.**
  sujet : Etude des propriétés fonctionnelles d'un réseau de neurones à codage temporel.
  Mars-Octobre 2008
  Utilisation d'un mécanisme d'apprentissage STDP pour apprendre des associations multimodales dans un réseau de cartes neuromimétiques impulsionnelles s'inspirant du principe de la CNFT.
- ▷ **M2R informatique (Nancy I) de Nicolas Beaufort. Co-encadrement avec O. Buffet.**
  sujet : Apprentissage optimiste et planification partielle pour un robot mobile.
  Février-Juin 2009
  Mise en œuvre d'un algorithme de construction incrémentale de modèle de l'environnement à partir de la mémorisation "paresseuse" des conséquences des actions d'un robot. Développement d'un algorithme de planification partielle qui s'appuie sur l'élaboration rapide d'une trajectoire "directe" qui est ensuite augmentée petit à petit en prenant en compte des états proches.
- ▷ **M2R informatique (Nancy I) de Louis Deflandre**
  sujet : Agent Epervier
  Février - Juin 2006.

Etude de l'apprentissage par renforcement sur la boucle sensori-motrice d'un agent. Le comportement de l'agent découle d'une perception active s'appuyant sur les champs neuroneaux continus.

- ▷ **DEA informatique (Nancy I) de Walid Tfaili**
  sujet : Robotique collective
  Février - Septembre 2003.
  Des robots mobiles dotés de comportements réactifs mettant en jeu satisfaction et altruisme coopèrent pour effectuer une tâche d'exploration collective.
- ▷ **DEA informatique (Nancy I) de Raghav Aras, co-encadrement avec François Charpillet et I. Chadès**
  sujet : Decentralized control in the pursuit domain
  Février - Juillet 2003.
  Approche hiérarchique et récursive des processus décisionnels de Markov qui permet d'appliquer les outils associés à des problèmes de grande taille ou continus.
- ▷ **DEA informatique (Nancy I) d'Olivier Buffet, Co-encadrement avec F. Charpillet**
  sujet : Jeux de Markov et apprentissage dans les systèmes multi-agents
  Avril - Août 2000
  Ces travaux sur une nouvelle méthode de conception des systèmes multi-agents ont débouché sur une publication à la conférence Agents'01.

Stages divers

- ▷ **Stage fin étude, Exia, de Jérôme Béchu**
  sujet : Traces d'éligibilité pour apprentissage par renforcement efficace sur Wifibot.
  Février-Aout 2008
  Améliorer l'efficacité de nos algorithmes d'apprentissage par renforcement embarqués par l'utilisation de traces d'éligibilité.
- ▷ **Stage fin étude, UTC, de François Rispal**
  sujet : Guidage collectif d'un robot dans un environnement multi-robotique.
  Septembre 2007 - Février 2008
  Définir, implémenter et expérimenter un algorithme permettant à deux robots dotés d'une caméra imprécise de guider un autre robot "aveugle".
- ▷ **Stage d'été, ESIAL, de Nicolas Beaufort**
  sujet : Apprentissage par renforcement efficace pour robot mobile.
  Juillet-Aout 2007
  Porter sur un Wifibot l'algorithme d'apprentissage par renforcement mis au point par W. Smart. La fonction de valeur est estimée en utilisant la régression locale pondérée.
- ▷ **Stage court, L2 Informatique, UHP-Nancy, de Renaud Fontana**
  sujet : Apprentissage par renforcement dans l'environnement Robocode.
  Juillet 2007
  Mettre en œuvre un 'bot' utilisant l'apprentissage par renforcement dans l'environnement 'Robocode' (http ://robocode.sourceforge.net).
- ▷ **Stage de Master 1 Sciences Cognitives de Eléonore Elias**
  sujet : Conceptions et analyse d'algorithmes pour jeux sans stratégies dominante, application à Alésia. 2006-2007

Proposer, implanter et étudier différents algorithmes pour permettre à des agents à apprendre à jouer à Alésia.

- ▷ **Stage d'été, ESIAL, de Julien LeGuen**
  sujet : Une librairie bas niveau pour Wifibot.
  Mai - Juillet 2006
  Conception d'une librairie pour interagir avec des Wifibot. Conception d'une pince. (voir https ://gforge.inria.fr/projects/wifibotlib).
- ▷ **Stage de fin d'Etude, MIAGE, de Cédric Bernier**
  sujet : Suivi de cible par un Wifibot.
  Juillet - Aout 2006.
  Augmenter la libraire développée par Julien LeGuen pour doter les Wifibots de la capacité de suivre des cibles données.
- ▷ **Stage de fin d'étude, IUP-SI Toulouse, d'Alexandre François**
  sujet : Apprentissage de modalités d'interaction
  Mars - Aout 2004
  Dans le cadre du projet européen Ozone, ce stage a vu la mise en œuvre d'un algorithme d'apprentissage par renforcement dans un espace d'état continu pour apprendre automatiquement les préférences des utilisateurs.
- ▷ **Stage de fin d'étude, IUP-SI Toulouse, de Pierre-Antoine Berreur**
  sujet : Mise en œuvre d'une plateforme multi-robotique
  Mars - Aout 2004
  Spécification et implémentation des routines de bas niveau d'une plateforme multi-robotique composée d'une caméra et de 10 robots "footballeurs" commandés par ondes hertziennes.
- ▷ **Stage IUT Charlemagne d'Aurélien Delaitre**
  sujet : Des 'bots intelligents sous NeL
  Juillet - Août 2002
- ▷ **Stage Initiation Recherche (Maîtrise Informatique Nancy I) de Nicolas Medoc et Joseph Razik**
  sujet : Une heuristique pour l'utilisation d'un contexte pour la prise de decision
  Février - Avril 2002

A.5 Enseignement

- ▷ **08/09 - Vacataire à l'Université Nancy II**
  10h de cours/TD sur l'apprentissage numérique pour le L3 de Science Cognitive.
- ▷ **07/08 - Vacataire à l'ESIAL, Nancy I**
  4h CO, 4h TD en "Introduction à l'Intelligence Artificielle", étudiants en 3ème année.
- ▷ **07/08 - Vacataire à l'Université Nancy II**
  10h de cours/TD sur l'apprentissage numérique pour le L3 de Science Cognitive.
- ▷ **06/07 - Vacataire à l'ESIAL, Nancy I**
  4h CO, 16h TD en "Introduction à l'Intelligence Artificielle", étudiants en 3ème année.
- ▷ **06/07 - Vacataire à l'Université Nancy II**
  10h de cours/TD sur l'apprentissage numérique pour le L3 de Science Cognitive.
- ▷ **05/06 - Vacataire à l'Université Nancy II**
  10h de cours/TD sur l'apprentissage numérique pour le L3 de Science Cognitive.
- ▷ **04/05 - Vacataire à l'Université Nancy II**

10h de cours sur les agents situés pour la Licence de Sciences Cognitives.

- ▷ **03/04 - Vacataire à l'Université Nancy II**
  10h de cours sur les agents situés pour la Licence de Sciences Cognitives.
- ▷ **02/03 - Vacataire à l'Université Nancy II**
  10h de cours sur les agents situés pour la Licence de Sciences Cognitives.
- ▷ **02/03 - Vacataire à l'Université Nancy I**
  3h de cours sur l'Apprentissage pour les DESS Ingéniérie du Logiciel.
- ▷ **01/02 - Vacataire à l'ESIAL, école d'Ingénieur en Informatique à Nancy**
  4h de cours, 26h de TD dans le module d'Intelligence Artificielle.
- ▷ **01/02 - Vacataire à l'Université Nancy I**
  3h de cours sur l'Apprentissage pour les DESS Ingéniérie du Logiciel, 26h de TD/TP pour les Licences d'Informatique en Technologie des Ordinateurs
- ▷ **01/02 - Vacataire à l'Université Nancy II**
  10h de cours sur l'Apprentissage pour les Maîtrises en Sciences Cognitives.
- ▷ **00/01 - ATER à l'ESIAL, école d'Ingénieur en Informatique à Nancy**
  20h de cours, 100h de TD, 50h de TP en Intelligence Artificielle, UNIX, Programmation orientée objet, Programmation en C, Technologie des Ordinateurs et Conception de Logiciels (UML/Merise).
- ▷ **99/00 - ATER à l'ESSTIN, école d'Ingénieur à Nancy.**
  28h de cours, 104h de TD et 30h de tutorat en programmation orientée objet, bases de données et UNIX.
- ▷ **94/99 - Vacataire à l'ENSAE, école d'Ingénieur en Aéronautique de Toulouse.**
  54h de cours, 62h de TD et 168h de TP en programmation orientée objet, intelligence artificielle, traitement du signal, mathématiques appliquées et projets longs des élèves.

A.6 Production logicielle

Depuis presque 3 ans, je me suis beaucoup investi dans le développement de l'activité robotique au Loria. Ces tâches de faible visibilité mais gourmandes en temps et en énergie ont permis l'acquisition d'une expérience et d'un savoir faire nécessaire à la mise en place d'expérimentations et de recherches en robotique. J'ai en particulier largement contribué à l'élaboration du contexte logiciel, embarqué et déporté, qui a permis d'assoir la recherche en robotique au Loria.

- ▷ **Bibliothèque bas niveau et driver URBI pour robots KheperaIII**
  J'ai largement contribué à l'écriture d'une bibliothèque bas niveau pour interagir avec les robots KheperaIII soit en utilisant le langage URBI soit par le biais de commandes envoyées par des sockets TCP/IP. Ces logiciels forment le cœur de nombreux travaux sur les robots kheperaIII du Loria et donc l'assise du dépot `loriakhep` sur la forge INRIA. *Disponibilité :* En téléchargement sur la gforge INRIA http://gforge.inria.fr/projects/loriakhep.
  Contributeurs : Nicolas Beaufort, Alain Dutech, Olivier Rochel
  Contact : Olivier.Rochel@loria.fr

- ▷ **Bibliothèque bas niveau pour les robots WifiBots**
  `wifibotlib` est une bibliothèque bas/moyen niveau pour controler les robots WifiBots. Cette bibliothèque permet d'interagir avec le Wifibot soit au niveau du hardware (con-

troler la vitesse des moteurs, lire les valeurs des capteurs) soit à un niveau un peu plus abstrait (avancer, tourner, s'arrêter).
Disponibilité : En téléchargement sur la gforge INRIA http://gforge.inria.fr/projects/wifibotlib.
Contributeurs : Nicolas Beaufort, Jérôme Béchu, Alain Dutech, Julien Le Guen, François Rispal, Olivier Rochel
Contact : Alain.Dutech@loria.fr

A.7 Collaborations

A.7.1 Collaborations académiques

▷ **Programme NeuroInformatique C.N.R.S. 2009 : *Adaptation & Action***
2009-2010.
La capacité pour un organisme d'acquérir plusieurs modèles ou représentations de l'environnement en parallèle peut être une stratégie d'adaptation efficace dans un environnement non-stationnaire. Chez le rat ou le primate, deux systèmes de contrôle de l'action instrumentale correspondent à des régions spécifiques du cortex préfrontal. Neurobiologistes et informaticiens trouvent ici un terrain commun pour identifier les critères auxquels obéissent ces systèmes et les processus d'arbitrage entre eux, qui pourraient dépendre de l'incertitude ou l'erreur associée à chacun. Sur la base de séquences d'actions réelles recueillies chez le rat intact ou lésé lors de procédures de dégradation de contingence, des modèles d'apprentissage par renforcement seront développés et évalués sur le plan formel afin d'évaluer les mécanismes permettant cette flexibilité comportementale.
Je participe à ce projet en tant que "spécialiste" de l'apprentissage par renforcement pour aider à modéliser les deux types d'apprentissage utilisés par les rats.
Partenaires : Loria-INRIA, Nancy (Maia et Cortex) ; Centre de Neurosciences Intégratives et Cognitives (CNIC), UMR 5228, Bordeaux (Fonctions exécutives : circuits neuronaux et développement) ; Supélec Metz (IMS).

▷ **ANR MAPS : Mappings, Adaptation, Plasticity and Spatial computation**
2008-2010.
Dans ce projet, par le biais de l'étude des propriétés d'une structure spécifique du cerveau (le Colliculus) envisagée sous plusieurs angles comme les Neurosciences, la Modélisation, la Psychologie Expérimentale, nous chercherons à mieux comprendre et à mieux modéliser les processus d'apprentissage et de traitement de l'information mis en oeuvre dans le cerveau, notamment au niveau des mécanismes temporels et spatiaux.
Je participe à ce projet en tant que "spécialiste" de l'apprentissage par renforcement. Un de mes objectifs est de proposer des algorithmes décentralisés d'apprentissage par renforcement compatibles avec les contraintes biologiques et notamment celles des neurosciences computationnelles.
Partenaires : Loria-INRIA, Nancy (Maia et Cortex) ; UMR Mouvement et Perception, Marseille ; Institut de Neurosciences Cognitives de la Méditerranée (INCM)-CNRS, Marseilles ; Laboratoire d'Informatique en Images et Systèmes d'information (LIRIS), Lyon.

▷ **Equipe Associée - Univ. of Massachusetts at Amherst, Resource-Bounded Reasoning Lab**
Je suis un membre actif de cette coopération renforcée entre notre équipe et l'équipe de Shlomo Zilbertein de UMass qui se traduit par le statut "d'équipe associée". Nous travaillons notamment sur la formalisation et l'analyse des processus décisionnels de Markov pour les sytèmes multi-agents. Dans ce cadre, avec Raghav Aras, nous avons conçu et validé l'algorithme de résolution exact de DEC-POMDP le plus rapide à l'heure actuelle.

▷ **GDR "Robotique et Apprentissage"**
Dans le GDR Robotique, au sein du Groupe de travail 4 "méthodologie", le sous groupe "apprentissage" vise à réunir l'ensemble des équipes françaises concernées par les méthodes d'apprentissage pour la robotique. La constitution d'un groupe de travail à part entière est en cours.

▷ **Groupe Processus Décisionnels de Markov et Intelligence Artificielle (PDMIA)**
Depuis 2001.
Je suis un des membres fondateurs d'un groupe de travail français sur les processus décisionnels de Markov dont le but est de favoriser des collaborations et de faire connaître la communauté française sur le plan international. Depuis 2001, des journées de travails sont organisées en France chaque année et, depuis 2006, une conférence sur le sujet (JFPDA). Par le biais de ce groupe, le noyau de chercheurs "expérimentés" dont je fais partie anime le recherche française dans le domaine, encadre les travaux des doctorants et participe à l'animation du domaine en dehors de la France. Ainsi, en 2008, le groupe a été l'un des moteurs de la "résurrection" du Workshop Européen d'Apprentissage par Renforcement (EWRL) qui a rassemblé toute la communauté internationnale du domaine à Lille en juillet 2008.

A.7.2 Transfert technologique

▷ **PROTEUS : Projet ITEA-01011a**
Octobre 2002 - Mars 2005.
Proteus vise à spécifier une plateforme pour faire de la e-maintenance industrielle. J'ai participé au WorkPackage 2 (Intelligence Artificielle) en tant que **co-leader**. Sur ce projet, l'équipe MAIA s'est intéressée plus particulièrement aux outils de l'Intelligence Artificielle qui peuvent être employés pour faire de la maintenance corrective et prédictive à distance, ce qui nous a permis de développer nos travaux sur l'apprentissage de modèle et sur la décision.
En tant que responsable du Workpackage 2, j'ai coordonné les travaux des différents partenaire et géré la rédaction des différents délivrables.

▷ **OZONE : Projet IST-2000-30026**
Septembre 2002 - Décembre 2004.
Au sein de ce projet d'*Intelligence Ambiante*, j'étais **responsable** de l'adaptation des services aux utilisateurs. Pour ce faire, nous avons développé un agent "intelligent" dont la fonctionnalité essentielle est de choisir le mode d'interaction (vocal, gestuel, etc) le plus adapté à l'interaction courante de l'utilisateur avec un programme client utilisant les services d'OZONE. Cela a été l'occasion pour l'équipe Maia de mettre en oeuvre ses acquis

en matière d'agent, d'apprentissage et de planification stochastique.

A.8 Responsabilités Collectives

Jury et comité de thèse

- ▷ **Jury de thèse de Raghav Aras**
  Thèse de l'Université Henry Poincaré de Nancy, soutenue le 23 octobre 2008 avec pour sujet : "Mathematical Programming Methods for Decentralized POMDPs".
- ▷ **Jury de thèse de Thomas Degris**
  Thèse de l'Université Paris 6, soutenue le 26 septembre 2006 avec pour sujet : "Apprentissage par renforcement dans les Processus de Décision Markoviens Factorisés".
- ▷ **Jury de thèse d'Olivier Buffet**
  Thèse de l'Université Henry Poincaré de Nancy, soutenue le 10 septembre 2003 avec pour sujet : "Une double approche modulaire de l'apprentissage par renforcement pour des agents intelligents adaptatifs".
- ▷ **Comité de thèse de Laurent Péret**
  Laurent Péret a effectué une thèse à l'INRA de Toulouse, et j'ai participé aux deux comités de thèse qui ont jalonné son travail (2001 et 2002). Ses travaux s'intéressent aux processus décisionnels de Markov de grande taille et à la planification pour les constellations de satellites.

Comité de lecture et Comité de Programme

- ▷ **Relecteur pour les revues suivantes**
  JAAMAS, RIA, IEEE Transactions on Systems, Man, and Cybernetics–Part C : Applications and Reviews, JESA.
- ▷ **AAAI-2008 : reviewer**
  La conférence américaine d'intelligence artificielle.
- ▷ **ICML-2010, 2008 : reviewer**
  La conférence sur le Machine Learning.
- ▷ **RFIA-2008 : Reviewer**
  Conférence française d'intelligence artificielle.
- ▷ **IJCAI-2007, 2005 : Reviewer**
  La Conférence généraliste internationale en Intelligence Artificielle.
- ▷ **JFPDA-2009, 2008, 2007, 2006 : Reviewer, Membre comité de Programme**
  La conférence qui a emané du groupe PDMIA. Rendez vous national de la communauté tournant autour de planification, décision et apprentissage.
- ▷ **PLMUDW-2006 : Reviewer**
  Workshop de la conférence ECAI-06 sur "Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds".
- ▷ **RJCIA-2003 : Reviewer et membre du comité de programme**
  Ces Rencontres des Jeunes Chercheurs en Intelligence Artificielles ont eu lieu à Laval en

juillet 2003.

- ▷ **EWRL-2008, 2003 : Reviewer et membre du comité de programme**
  Ce workshop international est *le* rendez vous des chercheurs de la communauté européenne sur l'apprentissage par renforcement.
- ▷ **PDMIA- 2005, 2004, 2003, 2002, 2001 : Reviewer et membre du comité de programme**
  Ateliers de travail du groupe sur les Processus Décisionnels de Markov et Intelligence Artificielle, précurseur des conférences JFPDA. Ce groupe regroupait tous les acteurs de la communauté française s'intéressant aux processus décisionnels de Markov dans le cadre de l'intelligence artificielle.
- ▷ **ECAI-2002 : Chairman ; 2002, 2008 : Reviewer**
  Lors de l'European Conference on Artificial Intelligence de juillet 2002 à Lyon, j'ai présidé la session "Agents".

Colloques et Congrès

- ▷ **NIPS 2005 - Neural Information Processing System Conference**
  J'ai été co-responsable d'un workshop intitulé 'Reinforcement Learning Benchmarks and Bake-offs II" qui vise à mettre en place des standarts et des benchmark dans le domaine de l'intelligence artificielle.
- ▷ **EWRL 2003 - European Workshop on Reinforcement Learning**
  J'ai été co-reponsable de l'organisation et du programme scientifique du Workshop EWRL-6 qui a eu lieu à Nancy au LORIA en septembre 2003. Ce workshop international est *le* rendez vous des chercheurs de la communauté européenne sur l'apprentissage par renforcement.
- ▷ **PDMIA - Processus Décisionnels de Markov et Intelligence Artificielle**
  Avec notamment F. Garcia de l'INRA Toulouse, P. Fabiani du CERT de Toulouse, F. Charpillet de l'INRIA Nancy, O. Sigaud de DASSAULT Paris et A.I. Mouaddib du Cril de Lens je suis un membre fondateur d'un groupe de travail français sur les processus décisionnels de Markov dont le but est de favoriser des collaborations et de faire connaître la communauté française sur le plan international.
  - J'ai notamment organisé une journée de travail à Nancy en 2002.
  - J'étais membre du comité de programme de la journée qui a eu lieu à Caen en juin 2003.

Au sein du laboratoire

- ▷ **Conseil de Laboratoire**
  J'ai été membre élu au conseil de laboratoire du Loria pour la période 2004-2008.
- ▷ **Commission de Spécialistes de Nancy 2**
  J'étais membre de la Commission de Spécialistes de Nancy 2 en 2006/2007.
- ▷ **AGOS**
  Depuis 2003, je suis membre du bureau du Comité de Gestion Local de l'Association de Gestion des Oeuvres Sociales de l'INRIA au sein du Loria. Je suis le secrétaire du CGL depuis 2005.
- ▷ **Groupe de Travail sur les Modèles Stochastiques**
  Je suis un des responsables d'un groupe de travail de l'équipe MAIA du Loria, et ce depuis

septembre 2000. Les notes de nos réunions sont disponibles sur le Web.
`http://maia.loria.fr/XXXXXX/GTMS`

- ▷ **Groupe Robotique du Loria**
  Je suis un des référents du développement et des recherches en robotique au Loria. Ce groupe rassemble les chercheurs, ingénieurs, doctorants et stagiaires dont l'activité touche à la robotique au sein du laboratoire.

A.9 Divers

- ▷ **Langues**
  Français (maternel), Anglais (courant), Allemand (scolaire), Arabe et Japonais (notions), Espagnol (débutant confirmé).
- ▷ **Titulaire du BAFA**
  Longtemps animateur de stages d'été pour l'association *ALTAIR* qui propose des vacances scientifiques pour les adolescents.
- ▷ **Sans oublier**
  Théâtre d'improvisation, sport (badminton, ski, basket), musique (batterie).

B

Annexes : Sélection d'Articles

Dans cette partie on trouvera les articles suivants :

- Apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés. Dutech A. et Samuelides. M. *Revue d'Intelligence Artificielle, RIA*, volume 17(4), pp 559–589, 2003.
  Cet article est en lien avec la section 3.4.

- Développement autonome des comportements de base d'un agent. Buffet O., Dutech A., Charpillet F. *Revue d'Intelligence Artificielle (RIA)*, volume 19(4-5), pp 603–632, 2005.
  Cet article est en lien avec la section 3.5.

- An investigation into Mathematical Programming for Finite Horizon Decentralized POMDPs. R. Aras and A. Dutech. *Journal of Artificial Intelligence Research*, Vol 37, pp 329-396, 2010.
  Cet article est en lien avec la section 4.2.

- Apprentissage par Renforcement et Théorie des Jeux pour la coordination de Systèmes Multi-Agents. A. Dutech, R. Aras et F. Charpillet. *Dans Colloque Africain sur la Recherche en Informatique - CARI 2006*, Cotonou, Bénin, 2006.
  Cet article est en lien avec la section 4.3.

- Shaping multi-agent systems with gradient reinforcement learning. Buffet O., Dutech A. et Charpillet F. *Journal of Autonomous agents and Multi-agent Systems*, Vol 15 :2, pp 197-220, 2007.
  Cet article est en lien avec la section 4.4.

Un algorithme d'apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés

Apprendre une extension sélective du passé

Alain Dutech* — **Manuel Samuelides****

* *LORIA - MAIA*
BP 239 - 54506 Vandoeuvre les Nancy
Alain.Dutech@loria.fr

** *Ecole Nationale de l'Aéronautique et de l'Espace*
10 avenue Edouard Belin
BP 4032 - 31055 Toulouse Cedex
samuelid@supaero.fr

RÉSUMÉ. *Nous présentons un nouvel algorithme qui contribue à étendre le formalisme de l'Apprentissage par Renforcement (RL) aux Processus Décisionnels Markoviens Partiellement Observés (POMDP). L'idée principale de notre méthode est de construire une extension d'état, appelée* observable exhaustif*, qui permet de définir un nouveau processus qui est alors markovien. Nous démontrons que résoudre ce nouveau processus, auquel on peut appliquer les techniques classiques de RL, apporte une solution optimale au POMDP original. Nous appliquons l'algorithme déduit de ce résultat sur plusieurs exemples pour en tester la validité et la robustesse.*

ABSTRACT. *We present a new algorithm that extends the Reinforcement Learning framework to Partially Observed Markov Decision Processes (POMDP). The main idea of our method is to build a state extension, called* exhaustive observable*, which allow us to define a next processus that is Markovian. We bring the proof that solving this new process, to which classical RL methods can be applied, brings an optimal solution to the original POMDP. We apply the algorithm built on that proof to several examples to test its validity and robustness.*

MOTS-CLÉS : *théorie de la décision, agent intelligent, apprentissage par renforcement, POMDP, extension d'état*

KEYWORDS: *decision theory, intelligent agent, reinforcement learning, POMDP, state extension*

RSTI - RIA. Volume 17 - n° 4/2003, pages 559 à 589

1. Introduction

Nous nous intéressons à la construction d'agents autonomes qui ont à résoudre des tâches complexes dans des environnements changeants et mal connus. Les agents interagissent avec leur environnement grâce à des capteurs et des actionneurs. Le problème qui se pose à eux est de choisir la "bonne" action en fonction de leur état interne et de leur perception du monde. Les méthodes d'apprentissage par renforcement sont particulièrement adaptées à ce genre de problème mais sont limitées, en théorie, aux tâches où l'état *complet* de l'environnement est connu par l'agent. En général les limitations physiques et en ressources de calcul font que l'environnement n'est que partiellement connu par l'agent. Cet article propose une réponse à la question : comment pouvons-nous utiliser l'apprentissage par renforcement pour résoudre des tâches dans un environnement partiellement connu ?

Les algorithmes d'apprentissage par renforcement sont des méthodes très attrayantes pour résoudre les problèmes de prise de décision car elles ne nécessitent qu'un signal de renforcement scalaire pour apprendre. Ainsi, un professeur passif est suffisant et, bien souvent, l'environnement lui même peut jouer ce rôle. Au travers d'algorithmes comme le Q-Learning [WAT 89] ou TD(λ) [SUT 88], il est possible de trouver des politiques d'actions optimales dans un environnement inconnu. Cependant, le formalisme mathématique sur lequel s'appuient ces algorithmes demande que l'environnement soit markovien pour qu'il y ait convergence : à chaque pas de temps, la connaissance de la *totalité* de l'état *présent* est nécessaire et suffisante pour choisir l'action optimale. Si cet état n'est que partiellement observé, les algorithmes classiques d'apprentissage par renforcement conduisent généralement à des solutions sous-optimales.

Plusieurs approches ont affronté ce problème. Aström [AST 65] utilise une extension d'état pour construire un nouveau problème markovien qui peut alors être résolu par des méthodes classiques, avec une difficulté accrue due au fait que ce nouvel état étendu est continu. Cassandra [CAS 98] et Littman [LIT 94] apportent des solutions pour prendre en compte cette explosion combinatoire. D'autres, comme Chapman et Kaelbling [CHA 91] utilisent des actions de perception pour différencier les diverses observations ambiguës.

Cependant, aucune de ces méthodes ne peut être appliquée quand l'environnement n'est que partiellement observé et que l'on ne connaît pas son modèle d'évolution, c'est-à-dire la structure de l'espace d'état et les probabilités de transition entre ces états. Jaakkola *et al.* [JAA 94b] ont proposé un algorithme pour ce type de problème, mais ne peuvent trouver que des solutions sous-optimales. Dans l'optique d'utiliser l'information contenue dans le passé du processus, Lin [LIN 92] utilise des réseaux de neurones récurrents pour construire des états internes stockant des faits pertinents du passé, mais il n'a rencontré qu'un succès limité.

Notre idée, comme dans [MCC 96], est de construire une extension d'état de dimension finie composée d'éléments non ambigus, ces éléments sont ce que nous appelons des trajectoires d'observation-action. Cette extension d'état permet alors de

chercher une politique de décision optimale en utilisant des algorithmes classiques. La section 2 présente le formalisme des processus décisionnels de Markov partiellement observés et introduit la notion de trajectoire d'observation-action avant de s'intéresser au formalisme mathématique sous-jacent à nos travaux. Notre algorithme est détaillé à la section 3 et des expériences pour le valider sont explicitées à la section 4. Nous comparons ensuite notre algorithme à d'autres approches similaires dans la ection 5. Nous poursuivons alors, en section 6, en passant en revue diverses améliorations possibles de notre algorithme avant de conclure (section 7).

2. Généralités et formalisme

2.1. *Q-Learning et POMDP*

Soit $(\mathcal{S}, \mathcal{A}, T, r)$ un **P**rocessus **D**écisionnel de **M**arkov (noté MDP) où $\mathcal{S}$ est un ensemble fini d'états, $\mathcal{A}$ un ensemble fini d'actions. Ce processus est markovien car les transitions d'états sont gouvernées par la fonction de transition $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow [0, 1]$ avec $P\{S_{t+1} = s' | A_t = a, S_t = s\} = T(s, a, s')$. r est une fonction de $\mathcal{S} \times \mathcal{A}$ dans $\mathbb{R}$ qui indique la **récompense** reçue par le système après chaque transition d'état. Une **politique** d'action π est une fonction des états vers les actions. Le problème est alors de trouver une politique d'action qui optimise une certaine fonction de la récompense, qu'on appelle l'**utilité**. Typiquement, l'utilité peut être la somme des récompenses sur un horizon fini, ou la somme pondérée de cette récompense sur un horizon infini ($\sum_{t=0}^{\infty} \gamma^t r_t$ où $\gamma \in [0, 1[$) ou encore la récompense reçue en moyenne à chaque transition.

Pour un MDP, il existe au moins une politique optimale qui est déterministe. On peut trouver cette politique optimale par des algorithmes dérivés de la programmation dynamique comme "*value iteration*" ou "*policy iteration*" quand T et r sont connues, [BEL 57], [PUT 94]. Dans les autres cas, on peut utiliser des méthodes d'apprentissage par renforcement, par exemple le *Q-Learning* de Watkins [WAT 89].

L'algorithme du Q-Learning estime l'utilité optimale de chaque couple (s, a) d'état-action par le biais d'une fonction Q. La convergence de l'algorithme est assurée si l'exploration de l'espace $\mathcal{S} \times \mathcal{A}$ est récurrente quand l'environnement est markovien, ainsi que le montre [JAA 94a]. Une fois que la fonction Q est connue, la politique optimale π^* est déterminée par : $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$. Dans ce cadre markovien, la convergence ne dépend pas de la stratégie d'exploration de l'espace d'état utilisée, sauf en ce qui concerne la vitesse de convergence, [THR 92]. La procédure standard est d'abord d'explorer l'environnement en choisissant des actions aléatoirement avant d'utiliser la politique optimale ainsi apprise, ce que nous appelons le *Q-Learning uniforme*. La plupart des algorithmes fusionnent lentement les stratégies d'exploration et d'exploitation au sein d'une même politique. C'est le cas du *Q-Learning de Boltzmann*. Ces deux algorithmes sont plus détaillés en annexe A.

Dans le cas qui nous intéresse, l'agent qui apprend n'a pas directement accès aux états du monde, il ne peut que les observer à travers des perceptions imparfaites. Cette

notion se formalise en ajoutant un espace d'observation $\mathcal{O}$ et une fonction de perception $f : \mathcal{S} \longrightarrow \mathcal{O}$ qui peut être stochastique. Ainsi, $\lambda = (\mathcal{S}, \mathcal{A}, T, r, \mathcal{O}, f)$ définit un **P**rocessus **D**écisionnel de **M**arkov **P**artiellement **O**bservé (noté POMDP). Le problème à résoudre maintenant est de trouver une politique optimale qui soit une fonction des observations et non plus des états, que l'agent ne connaît pas. Nous parlerons alors de **politique adaptée** quand elle sera fonction des observations.

Sans perdre en généralité, nous allons considérer que la fonction d'observation f est déterministe car, pour tout POMDP avec une fonction d'observation stochastique, il est possible de construire un POMDP équivalent avec plus d'états mais une fonction d'observation déterministe.

Comme l'a montré [SIN 94], les algorithmes classiques d'apprentissage par renforcement ne peuvent construire des politiques *adaptées* optimales car ils construisent des politiques markoviennes et déterministes. Or, par des exemples, Sing [SIN 94] montre que la politique optimale, fonction des états, peut être arbitrairement meilleure que n'importe quelle politique stochastique adaptée. De même, ces dernières peuvent à leur tour être arbitrairement meilleures que n'importe quelle politique adaptée déterministe. L'unique solution pour atteindre l'optimalité est d'utiliser des politiques adaptées qui soient *non markoviennes*.

2.2. *Q-Learning adapté*

Dans certains cas, il peut être suffisant de trouver des politiques sous-optimales, surtout quand ce sont les seules que l'on peut obtenir en termes de temps de calcul. En fait, nous avons vu de nombreux travaux qui utilisent des algorithmes de Q-Learning classiques sur des environnements non markoviens et qui obtiennent des résultats intéressants. Mais, dans le cas général, comme le montre [SIN 94], le Q-Learning ne peut pas converger pour des problèmes non markoviens.

En théorie, la convergence du Q-Learning pour des POMDP n'est possible que si on utilise une politique *fixée* pendant la phase d'exploration de l'environnement. Cette politique peut évidemment être stochastique, mais elle doit être fixée : pour un état donné, la probabilité de choisir une action doit être constante. Ce n'est généralement pas le cas pour les méthodes classiques. Si la politique d'exploration est fixée, l'algorithme du Q-Learning (que nous appelons alors *Q-Learning adapté*) converge vers l'estimation de l' *utilité de la politique fixée utilisée*. On ne peut alors être sûr de trouver l'utilité optimale.

Cependant, [JAA 94b] montre comment il est possible d'améliorer la politique d'exploration fixée à partir de son utilité. L'itération de ce procédé est au coeur de l'algorithme d'apprentissage par renforcement qu'il propose. Cet algorithme est assuré de converger vers un maximum local de l'utilité et utilise des politiques stochastiques. A notre connaissance, c'est le seul algorithme d'apprentissage par renforcement pour les POMDP qui s'appuie sur des résultats mathématiques.

Comme l'article de Jaakkola *et al.* [JAA 94b] ne donne pas de résultats expérimentaux, nous l'avons testé sur des exemples simples. Ces expériences sont décrites dans [DUT 99] et ont montré que cet algorithme avait des performances similaires à celle du Q-Learning de Boltzmann adapté aux seules observations. De plus ce dernier semble plus robuste et rester moins souvent bloqué dans des maximums locaux. Cependant, il faudrait tester et comparer ces deux algorithmes sur des exemples plus complexes et moins académiques avant de prononcer des conclusions définitives.

2.3. *Les états estimés d'Aström*

Quand on connaît le **modèle d'évolution du processus**, c'est-à-dire les transitions T et la récompense r, il est possible, en théorie, de trouver une politique adaptée optimale pour un POMDP. Aström [AST 65] propose en effet d'utiliser des états étendus qui sont en fait des distributions de probabilité sur l'espace des états $\mathcal{S}$. Il appelle ces états étendus des **états estimés**, ou *belief states* en anglais. Un état estimé est défini par : $(s) = P\{S_t = s\}$. Le principe de la méthode d'Aström est que si on connaît l'état estimé $_t$, l'observation O_t et l'action A_t choisie à l'instant t, alors il est possible de déterminer le nouvel état estimé $_{t+1}$ à l'instant $t+1$ par une estimation bayésienne. L'extension d'état permet ainsi de construire un nouveau processus décisionnel *markovien* que l'on peut alors résoudre avec des techniques classiques.

Le problème est que le nouvel espace d'état étendu est potentiellement infini et continu ! Il est alors quasiment impossible de résoudre le MDP associé à cet espace d'état et l'on doit souvent se résoudre à chercher des solutions approximatives. Ainsi [LIT 94] ou [CAS 98] étudient des algorithmes d'approximation de l'utilité de politiques définies sur les états estimés en utilisant des fonctions paramétriques qui doivent être apprises.

La méthode d'Aström n'est de toute façon pas directement applicable à notre cadre puisque nous ne connaissons pas le modèle d'évolution du POMDP que nous essayons de résoudre. Néanmoins, l'approche théorique proposée par Aström est fort intéressante. En effet, elle montre que les états estimés sont une sorte de résumé exhaustif de tout le passé du processus. Or, parfois, une telle connaissance de tout le passé n'est pas nécessaire pour avoir assez d'information sur le présent pour être capable de choisir l'action optimale. Nous avons décidé de n'utiliser le passé du processus que là où il est vraiment utile, ce qui permet d'utiliser une extension d'état avec un nombre fini d'éléments. Evidemment, ce n'est réalisable que pour une sous-classe des POMDP, comme nous l'expliquons ci-dessous.

2.4. *POMDP d'ordre N*

Commençons par un petit exemple. Supposons que nous voulons faire un paquet cadeau. La démarche est simple : ouvrir la boîte, mettre le cadeau dans la boîte, fermer la boîte, emballer la boîte dans du papier cadeau. Si nous ne pouvons observer que

l'état de la boîte (*fermée*, *vide*, *pleineEtOuverte* ou *emballée*), nous sommes face à un POMDP. En effet, quand la boîte est fermée, nous ne pouvons plus savoir si le cadeau est à l'intérieur ou non, sauf si nous avons une mémoire du passé. Dans ce cas seulement, nous pouvons savoir si nous avons mis le cadeau à l'intérieur précédemment. Par contre, quand la boîte est ouverte, il n'y a pas d'ambiguïté sur l'état de l'environnement et nous pouvons choisir l'action optimale en fonction de la présence ou non du cadeau dans la boîte. Ainsi, si nous utilisons les états étendus $_{\text{boite}} = \{$ (*pleineEtOuverte*), (*vide*), (*emballée*), (*fermée*, *actFermer*, *pleine*), (*fermée*, *actFermer*, *vide*)$\}$ où les événements perceptifs et actifs sont listés du plus récent au plus ancien, nous avons à chaque fois assez d'information pour choisir l'action optimale. La figure 1 illustre cet exemple qui nous permet de montrer que la mémoire de tout le passé n'est pas nécessaire pour résoudre ce POMDP. Dans certains cas, par exemple quand nous observons que la boîte est vide, nous n'avons même pas besoin de mémoire du tout.

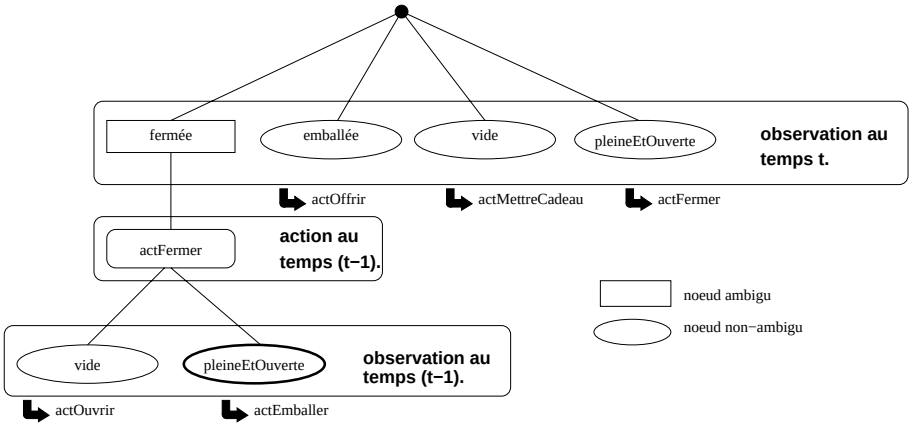


Figure 1. *Espace d'états étendu pour l'emballage de cadeau. Cet espace peut se mettre sous la forme d'un arbre où chaque feuille représente un état étendu non ambigu. Ainsi, la feuille que nous avons particularisée par un trait plus épais correspond à l'état étendu* j'observe que la boite est fermée alors que je viens de la fermer et qu'elle était pleine *que nous notons (fermée, actFermer, pleine). A chacun de ces états étendus, nous pouvons associer une action optimale, ce qui n'est pas le cas de l'observation* fermée *qui est ambiguë.*

Nous allons maintenant formaliser ces notions. Nous appellerons **Trajectoire d'Etat-Action** (notée TEA) une séquence d'états et d'actions. Ainsi, la TEA courante du processus s'écrit :

$$\Sigma_t = (s_t, a_{t-1}, s_{t-1}, \ldots, s_1, a_0, s_0)$$

De la même manière, nous appellerons **Trajectoire d'Observation-Action** (ou encore TOA) une séquence d'observations et d'actions, comme par exemple $\omega = (o_t, a_{t-1}, o_{t-1}, \ldots, a_{t-k}, o_{t-k})$. L'**ordre** d'une trajectoire est simplement le nombre d'observations ou d'états qu'elle contient. Ainsi, la TOA ω précédente est d'ordre $k+1$. Pour chaque TEA $\sigma = (s_i, a_{i-1}, \ldots, s_0)$ on peut obtenir une TOA ω en utilisant la fonction d'observation f par $\omega = f(\sigma) = (f(s_i), a_{i-1}, \ldots, f(s_0))$.

La **concaténation** de deux trajectoires $\omega = (o_i, \ldots, a_{i-k}, o_{i-k})$ et $\omega' = (o'_j, \ldots, a'_{j-l}, o'_{j-l})$ où $o_{i-k} = o'_j$ est définie par :

$$\omega \cdot \omega' = (o_i, a_{i-1}, \ldots, o_{i-k}, a'_{j-1}, o'_{j-1}, \ldots, o'_{j-l})$$

La concaténation induit un *ordre partiel* sur l'ensemble des TOA défini par :

$$(\omega_1 \quad \omega_2) \Leftrightarrow (\ \omega' | \omega_2 = \omega_1 \cdot \omega')$$

Dans ce cas, on dit que ω_1 est un **minorant** de ω_2.

Si l'on considère un ensemble $\mathcal{F}$ de TOA, comme l'ensemble $_{\text{boite}}$ de l'exemple de la boite, on définit un opérateur de **projection** noté $proj_{\mathcal{F}}$. Ainsi, la projection d'une TOA ω sur l'ensemble $\mathcal{F}$ est le plus grand minorant (la borne supérieure) de ω appartenant à $\mathcal{F}$, s'il existe. Sinon, l'image de ω par la projection est l'ensemble vide . Si l'on reprend l'exemple de la boite, la projection de $(pleineEtOuverte,$ $act \quad ettreCadeau, vide)$ (resp. $(fermée)$) sur l'ensemble $_{\text{boite}}$ est $(pleineEtOuverte)$ (resp.).

La fonction $proj_{\mathcal{F}} \circ f$ est un **observable** du processus. Par abus de langage, nous dirons aussi que l'ensemble $\mathcal{F}$ est un observable du POMDP. L'ordre de l'observable est l'ordre de la plus grande trajectoire de l'observable.

Un observable définit une relation d'**équivalence** sur $\mathcal{S}$ que nous notons "$\sim$". Quels que soient les états s et s' de $\mathcal{S}$ et toute trajectoire d'observation-action ω d'ordre supérieur à celui de $\mathcal{F}$, nous écrivons que :

$$s \sim s' \Leftrightarrow (proj_{\mathcal{F}}(f(s), a, \omega) = proj_{\mathcal{F}}(f(s'), a, \omega))$$

Nous pouvons maintenant aborder le point clef de notre méthode, l'**exhaustivité**. Un observable $\mathcal{F}$ est exhaustif quand il contient assez d'information pour pouvoir prédire le comportement du POMDP. Nous le traduisons par :

Définition 1. *Un observable $\mathcal{F}$ d'ordre n est exhaustif pour un processus décisionnel de Markov partiellement observé s'il vérifie :*
a) pour toute TOA ω d'ordre supérieur à n, on a $proj_{\mathcal{F}}(\omega) \neq$.
b) pour toute TEA Σ_t, pour tout a de $\mathcal{A}$, tout $t \geq n$, nous avons $P\{s_{t+1}|a, \Sigma_t\} = P\{s_{t+1}|a, proj_{\mathcal{F}}(f(\Sigma_t))\}$
c) la récompense r est $\mathcal{F}$-adaptée : $(s \sim s') \Longrightarrow (r(s,a) = r(s',a))$

La première condition permet d'être sûr que nous pourrons, à tout moment, associer un élément de l'observable à la trajectoire d'état-action du processus. La deuxième

condition dit que quand on connaît l'élément de l'observable on en sait assez sur le POMDP pour choisir une action optimale. La troisième condition est nécessaire pour prouver les théorèmes qui vont suivre. Cette définition est très proche de celles des processus markoviens d'ordre n. On rajoute ici la notion de classe d'équivalence pour la récompense. Et nous rappelons que, en pratique, l'observable $\mathcal{F}$ peut contenir des éléments d'ordre très inférieur à n.

Un POMDP n'admet pas forcément un observable exhaustif fini, c'est pourquoi nous allons seulement considérer la classe des **POMDP d'ordre N**.

Définition 2. *Un POMDP d'ordre N admet un observable exhaustif d'ordre N.*

Pour cette classe de problèmes, le point crucial est de trouver l'observable exhaustif. En effet, comme le montrent les deux théorèmes suivants que nous avons démontrés, si nous trouvons un tel observable, nous pouvons alors construire une extension d'état qui définit un MDP dont la politique optimale sera une politique optimale adaptée pour le POMDP.

Le premier théorème montre comment on peut construire l'extension d'état qui génère alors un MDP.

Théorème 1. *Si λ est un processus décisionnel de Markov partiellement observé qui admet un observable exhaustif $\mathcal{F}$ alors le problème de décision étendu défini par $(\hat{S}_t = proj_{\mathcal{F}}[f(\Sigma_t)])_{t \geq n}$ est un processus décisionnel de Markov $\hat{\lambda} = (\hat{\mathcal{S}}, \hat{\mathcal{A}}, \hat{T}, \hat{r})$ où :*
a) $\hat{\mathcal{S}} = \mathcal{F}$, $\hat{\mathcal{A}} = \mathcal{A}$
b) pour tout $(\hat{s} = f(s).\omega) \in \hat{\mathcal{S}}$ nous avons
$\hat{r}(\hat{s}, a) = r(s, a)$
c) $\hat{T}(\hat{s}, a, \hat{s}') = P\{\hat{S}_{t+1} = \hat{s}' | A_t = a, \hat{S}_t = \hat{s}\}$

Le deuxième théorème montre que la récompense obtenue en utilisant la politique optimale du problème de Markov étendu est la même que celle que nous obtiendrions en ayant accès aux états du POMDP.

Théorème 2. *La politique optimale $\hat{\pi}^*$ du processus décisionnel de Markov étendu $\hat{\lambda}$ définit en fait une politique* adaptée *pour le processus décisionnel de Markov partiellement observé λ. C'est-à-dire que la récompense reçue en utilisant $\hat{\pi}^*$ est la même que celle reçue en utilisant la politique non adaptée optimale π^*.*

La démonstration de ces deux théorèmes est donnée en annexe B.

3. Algorithme de construction d'un observable exhaustif

Le principe de notre algorithme d'apprentissage par renforcement pour les POMDP est de construire un observable exhaustif qui nous permettra alors de trouver une politique adaptée optimale. La construction de cet observable sera incrémentale : les

éléments ambigus seront peu à peu remplacés par des trajectoires non ambiguës. Dans un premier temps, nous expliquons comment nous détectons les TOA ambiguës, ensuite nous explicitons la construction incrémentale de l'observable. Nous abordons alors le problème du critère d'arrêt de l'algorithme. Nous précisons enfin les détails de l'algorithme.

3.1. *Détecter les ambiguïtés*

Pour pouvoir construire incrémentalement l'observable, nous avons besoin de pouvoir les éléments ambigus d'un observable courant. Pour cela nous nous appuyons sur une heuristique qui découle des constatations et des motivations suivantes.

3.1.1. *Convergence plus lente*

Un observable donné permet de définir une extension d'état induisant un processus décisionnel qui peut ne pas être markovien, surtout si l'observable n'est pas exhaustif. Cependant, on peut quand même calculer l'utilité de chaque couple (*élément de l'observable, action*) en utilisant un algorithme de Q-Learning adapté (voir section 2.2). Si la politique utilisée pendant la phase d'exploration est fixée, cet algorithme va converger. De plus, nous avons noté que la vitesse de convergence était plus rapide pour les éléments non ambigus que pour les éléments ambigus.

Nous exploitons ce fait à travers les variations de la fonction $Q(\omega, a)$ où ω est un élément de l'observable. Plus l'amplitude de la variation est grande, plus l'élément a des chances d'être ambigu.

3.1.2. *Nombre de mises à jour*

Le nombre de mises à jour de la fonction Q pour un couple (ω, a) influence aussi les variations de Q par l'intermédiaire du coefficient α d'apprentissage (voir annexe A) qui, décroissant avec le nombre de mises à jour, fait diminuer les variations de Q. Il nous faut alors tenir compte de ce nombre de mises à jour pour la détection d'éléments ambigus car certains éléments mis peu souvent à jour auront des fortes variations.

De plus, il est assez fréquent que, en utilisant une politique d'exploration aléatoire uniforme, les éléments ambigus soient visités plus souvent que les éléments non ambigus. Cela vient du fait que ces éléments ambigus sont associés à plusieurs états du MDP sous-jacent au POMDP. Les éléments non ambigus seront plus probablement associés à un seul état sous-jacent.

Dans le cas général, un élément a plus de chances d'être ambigu quand son nombre de mise à jour est grand.

3.1.3. *Ambiguïté dans les actions*

Si l'action optimale pour un élément ambigu est clairement définie, bien qu'il soit ambigu, nous n'allons pas forcément chercher à lever cette ambiguïté. Inversement,

quand les valeurs de la fonction Q d'un élément sont très proches pour les différentes actions possibles, l'action optimale n'est pas clairement établie. Dans ce cas, il peut être intéressant de traiter l'élément comme étant ambigu pour le remplacer par des éléments qui, apportant plus d'information sur le passé, permettront peut-être de choisir plus clairement l'action optimale.

C'est pourquoi nous considérons qu'un élément dont les deux plus grandes valeurs de Q sont très proches est un élément ambigu potentiel.

3.1.4. *En bref*

Chaque élément de l'observable est évalué en fonction de ces trois critères. Pour cela nous classons les éléments selon chacun de ces critères et considérons la somme des classements selon ces trois critères. Les éléments les plus ambigus sont ceux dont la somme est la plus basse. Cette heuristique, bien que fort simple, donne de bons résultats, comme nous le montrons dans la section 4.

3.2. *Construire l'observable*

La construction de l'observable est itérative. Au départ, notre observable $\mathcal{F}$ est constitué de l'ensemble des observations du POMDP.

Comme nous venons de l'expliquer, en utilisant un algorithme de Q-Learning uniforme adapté à cet observable, nous pouvons en détecter les éléments les plus ambigus. Supposons que ω soit un de ces éléments. Alors, nous remplaçons ω dans $\mathcal{F}$ par $|\mathcal{A}| \times |\mathcal{O}|$ éléments de la forme (ω, a, o) où a appartient à $\mathcal{A}$ et o appartient à $\mathcal{O}$. Nous augmentons ainsi notre observable $\mathcal{F}$. Nous itérons ensuite le procédé jusqu'à ce que l'observable soit exhaustif.

3.3. *Critère d'arrêt*

En fait, notre heuristique ne nous permet pas de détecter les éléments ambigus d'un observable, mais seulement les éléments les *plus* ambigus. Dès lors, il n'est pas possible de décider si un observable est exhaustif ou pas. Nous avons alors mis en place un critère d'arrêt pour notre algorithme.

Chaque nouvel observable construit est évalué et nous arrêtons notre algorithme quand les performances n'augmentent plus. Pour évaluer un observable, nous estimons l'utilité moyenne de la meilleure politique construite sur cet observable en utilisant le Q-Learning de Boltzman. L'utilisation de cet algorithme alors que l'observable n'est pas forcément exhaustif est justifiée par sa robustesse (voir section 2.2 et [DUT 99]). Quant au choix de l'utilité moyenne comme estimation de la qualité d'une politique, Singh *et al.* [SIN 94] montrent que c'est un critère particulièrement adapté aux POMDP.

Nous imposons aussi une limite au nombre d'éléments que peut contenir un observable pour éviter que l'observable ne grossisse indéfiniment.

3.4. *L'algorithme en détail*

Comme nous venons de l'expliquer, notre algorithme est composé d'une succession de deux parties : une phase d'exploration et une phase d'évaluation de performance. Nous allons ici donner le détail de l'algorithme, pour cela nous introduisons les notations suivantes. Comme précédemment, nous considérons un POMDP $(\mathcal{S}, \mathcal{A}, T, r, \mathcal{O}, f)$. A chaque instant t, le processus est dans l'état s_t, l'agent perçoit l'observation o_t, il choisit d'effectuer l'action a_t ce qui lui permet de recevoir la récompense r_t.

Une trajectoire d'observation-action d'un observable $\mathcal{F}$ sera notée ω. Des algorithmes de Q-Learning seront utilisés pour estimer la fonction Q associée à chaque couple (ω, a). Nous en noterons $Q_n(\omega, a)$ la $n^{ème}$ mise à jour. $N(\omega, a)$ est le nombre de fois où $Q_n(\omega, a)$ a été mis à jour, et $\Delta Q_n(\omega_t, a_t) = (Q_n(\omega_t, a_t) - Q_{n-1}(\omega_t, a_t))$.

Les paramètres de l'algorithme sont :

– N_{ambigu} : le nombre d'éléments ambigus pris en compte à chaque étape de la construction de l'observable.

– T_{dec} et T_{start} : pour le Q-Learning de Boltzman.

– n_{explor} : nombre d'itérations pour l'exploration du premier observable.

– N_{max} : taille maximale de l'observable.

1) **Initialisation**
$\mathcal{F} = \mathcal{O}$

2) **Exploration**
En utilisant le Q-Learning uniforme sur $\mathcal{F}$ (voir Annexe A), nous mettons à jour $Q_n(\omega, a)$, $N(\omega, a)$ et $\Delta Q_n(\omega_t, a_t)$. Cette exploration est faite pendant n_{explor} itérations, ce nombre augmente avec la taille de l'observable.

3) **Détection des ambiguïtés**
Comme indiqué à la section 3.1, nous utilisons une heuristique. Les éléments sont donc classés selon trois critères avant qu'on ne choisisse les N_{ambigu} éléments les plus ambigus (plus petites somme des classements) qui forment l'ensemble . Les trois critères sont :

- *Rapidité de convergence.* Les ω sont classés du plus grand $\Delta Q_n(\omega_t, a_t)$ moyen au plus petit.

- *Nombre de mises à jour.* Les ω sont classés du plus grand $N(\omega, a)$ au plus petit.

- *Pas d'action claire.* Les ω sont classés par ordre croissant de différence entre la valeur Q de l'action préférée et de la valeur de Q de la seconde action préférée.

4) **Performance**
En utilisant le Q-Learning de Boltzmann (voir annexe A), nous évaluons les performances de l'observable courant par le biais de la récompense moyenne reçue en suivant la politique adaptée ainsi déterminée. Si les performances n'augmentent plus ou si la taille limite de l'observable est atteinte, nous allons à l'étape 6.

5) **Augmenter l'observable**
Chaque élément ω de est remplacé par tous les éléments (ω, a, o) où a décrit $\mathcal{A}$ et o décrit $\mathcal{O}$. On retourne à l'étape 2.

6) **Stop**

4. Résultats expérimentaux

Nous avons testé notre algorithme dans des environnements statiques et dynamiques pour mesurer ses performances et en apprendre plus sur son comportement. Tous ces problèmes sont des POMDP mais le dernier sort du cadre théorique d'application de notre algorithme car il n'existe pas d'observable exhaustif fini. Notre algorithme a de bonnes performances et reste robuste même dans des conditions difficiles.

4.1. *Environnement statique*

Le premier problème auquel nous nous intéressons est le fameux problème du *cheese maze*, voir [MCC 95] ou [CAS 98]. Dans cet environnement discret simulé dépeint à la figure 2, une souris doit trouver son chemin à travers un labyrinthe pour aller jusqu'au fromage. La souris est l'agent apprenant. A chaque pas de temps, la souris doit choisir entre quatre actions : *Haut*, *Gauche*, *Bas* ou *Droite*. Les différentes positions de la souris dans son environnement forment l'ensemble des états (11 états différents).

La souris ne perçoit pas directement son état, mais observe simplement la présence de murs autour d'elle. Ainsi, dans l'exemple de la figure 2, la souris ne perçoit que les murs qui sont à sa gauche et à sa droite. Dès lors, dans cette même figure, les cases 2 et 4 semblent identiques pour la souris. Il en va de même pour les cases 6, 7 et 8. La récompense donnée à la souris est de -1.0 si la souris rentre dans un mur, de -0.1 si elle bouge et de +5.0 si elle trouve le fromage. Quand elle trouve le fromage, la souris est transportée automatiquement dans une autre case du labyrinthe et doit retourner vers le fromage.

Cette tâche devrait être assez facilement résolue par la souris. Elle doit apprendre que pour les cases ambiguës, une mémoire très limitée du passé (la dernière observation) est nécessaire. Par exemple, dans une case hachurée vers la gauche (observation E), si la souris vient de la case 3 elle doit alors descendre sinon elle doit monter. Ce problème est un processus décisionnel markovien partiellement observé d'ordre 2.

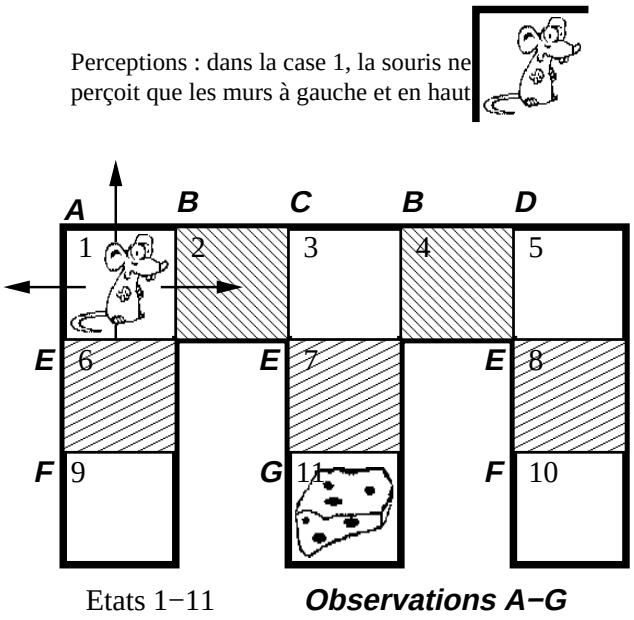


Figure 2. *La souris et le fromage. La souris doit choisir entre 4 actions pour aller au fromage. Mais elle n'a qu'une connaissance partielle de son état car elle ne perçoit que les murs qui l'entourent. Dès lors, les cases 6, 7 et 8 lui semblent identiques et forment l'observation E. La souris est punie quand elle heurte un mur et récompensée quand elle trouve le fromage.*

4.1.1. *Actions déterministes*

Nous avons d'abord vérifié que des techniques classiques d'apprentissage par renforcement ne pouvaient pas trouver de politique optimale pour ce problème. Pour cela nous avons utilisé le Q-Learning de Boltzmann pour essayer de trouver une politique adaptée markovienne. Nous avons fait tourner cet algorithme pendant 18000 itérations, le coefficient d'apprentissage α_t décroissant comme $\frac{1}{t}$. Pour les paramètres de la décroissance de la température qui gouverne les distributions de probabilité de Boltzmann sur les actions, nous avons choisi $T_{start} = 5,0$ et $T_{dec} = 0,999$. Avec ces paramètres, l'action ayant la plus grande Q-valeur est choisie 99% du temps après 5000 itérations.

Ensuite, nous avons utilisé notre algorithme. La longueur de la première période d'exploration est de 4000 itérations. Après cette phase, les trois actions les plus ambiguës sont choisies pour former le nouvel observable. Avant d'explorer ce nouvel observable pendant 6000 itérations, nous mesurons les performances du premier observable avec un Q-Learning de Boltzmann ($T_{start} = 5,0$, $T_{dec} = 0,995$) pendant 2500 itérations. Ensuite le second observable est évalué à son tour ($T_{start} = 5,0$, $T_{dec} = 0,995$).

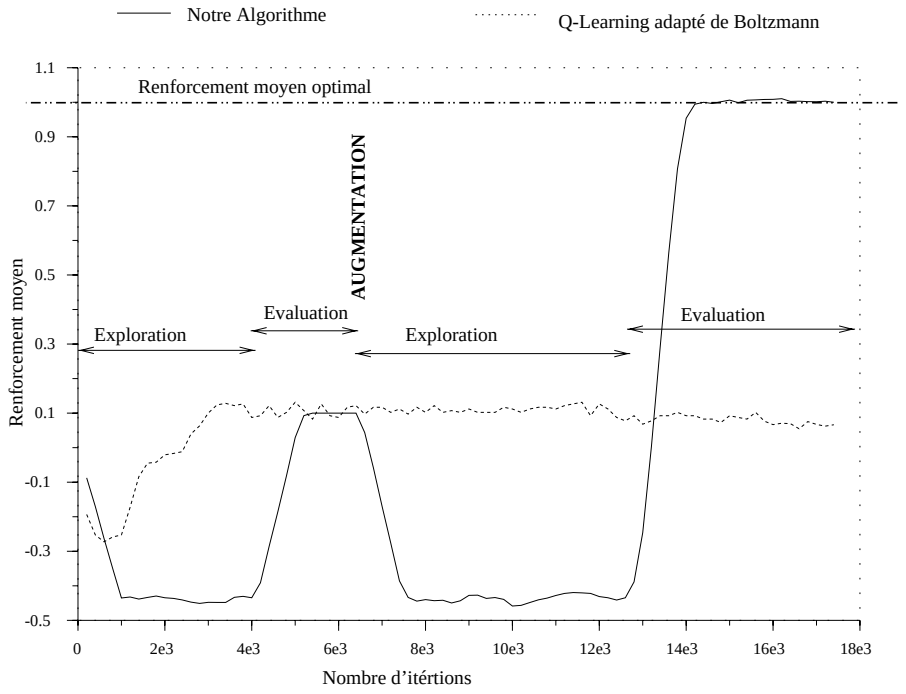


Figure 3. *Performances comparées. Le Q-Learning classique, qui est limité à des politiques markoviennes adaptées, ne peut pas trouver une politique optimale. Dans un premier temps, notre algorithme trouve la même politique adaptée mais, ensuite, il utilise des éléments du second ordre pour converger vers une politique optimale.*

La figure 3 montre les performances de ces deux algorithmes sous la forme d'un graphe donnant l'évolution de la récompense moyenne en fonction du nombre d'itérations. On y voit clairement que la politique markovienne reste sous-optimale et limitée à une récompense moyenne de $0,1$ tandis que notre algorithme converge vers une politique non-markovienne qui est optimale. Ces résultats sont les moyennes des résultats obtenus sur 10 simulations.

Comme nous l'espérions, après la première phase d'exploration, les observations les plus ambiguës sont E et B, ensuite viennent soit A soit D. La longueur de la période d'exploration joue un rôle important car nous avons vu qu'avec une période d'exploration de moins de 100 itérations, les observations E et B ne sont pas toujours considérées comme les plus ambiguës. Ce rôle est partagé par la valeur initiale et la vitesse de décroissance du coefficient d'apprentissage. Cela est dû au fait que, avec un faible nombre d'itérations, la fonction Q n'a pas toujours le temps de converger suffisamment.

4.1.2. *Actions stochastiques*

Dans une deuxième expérience, nous avons utilisé des actions stochastiques. Quand la souris choisit l'action *Haut* par exemple, elle ne va pas toujours vers le haut. En fait, chacune des actions qui n'a pas été choisie peut tout de même être exécutée avec une probabilité de q, ce qui fait que la probabilité d'effectuer l'action choisie est de $(1 - 3q)$. Cette difficulté supplémentaire ne devrait pas empêcher la souris de trouver le fromage, sauf si q est trop grand.

Nous avons d'abord testé notre algorithme avec un bruit $q = 0,1$. La première période d'exploration est de 15000 itérations et les 3 observations les plus ambiguës sont choisies pour augmenter l'observable. Pour la première phase d'évaluation nous avons employé un Q-Learning de Boltzmann avec les paramètres suivants : $T_{start} = 5,0$, $T_{dec} = 0,995$, 15000 itérations. Ensuite, le deuxième observable est exploré pendant 40000 itérations avant d'être évalué à son tour ($T_{start} = 5,0$, $T_{dec} = 0,9998$, 25000 itérations). La figure 4 donne les résultats moyens sur 20 simulations.

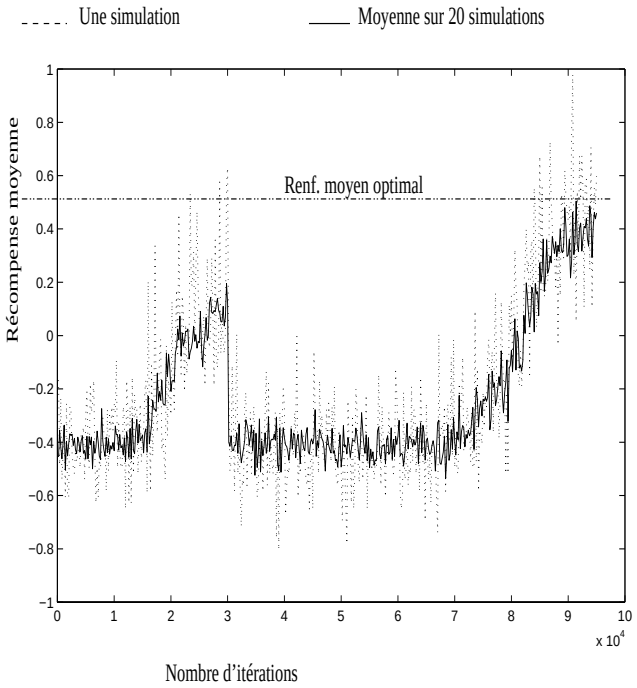


Figure 4. *Avec du bruit. Ce graphe présente les résultats de notre algorithme moyennés sur 20 simulations quand les actions sont bruitées ($q = 0,1$)*

Il est évident que la récompense optimale est inférieure à celle du cas précédent. Les observations B et E sont toujours considérées comme les plus ambiguës. Pour être plus systématique et parce que nous pensions que notre algorithme ne se comporterait pas aussi bien en présence de bruit plus important, nous avons lancé une série

d'expérience avec des q croissants. A partir d'une valeur initiale de $0,05$, q croit de $0,05$ en $0,05$ jusqu'à une valeur limite de $0,25$. Pour chacune de ces valeurs nous lançons 20 simulations et regardons les observations les plus ambiguës et la valeurs de la récompense moyenne obtenue. Ces résultats sont résumés dans le tableau 1.

Bruit	Réc. Moy.	Plus Ambiguë	Suivante	Suivante
$0,05$	$0,61$	E	B,AD	G,AD
$0,10$	$0,57$	E	B,ADG	B,G
$0,15$	$0,12$	E	B,AD,G,F	B,G,AD,F
$0,20$	$-0,19$	E	A,F,B	F,B,D,C
$0,25$	$-0,39$	F,E,A	E,AD,F	F,AD,E

Tableau 1. *Bruit croissant. Pour chaque niveau de bruit q, nous donnons la récompense moyenne reçue sur 20 simulations ainsi que les 3 observations les plus ambiguës détectées par notre algorithme. Quand les observations sont séparées par des virgules, comme par exemple E,A,B, cela indique que E fut le choix le plus fréquent, puis A puis B.*

Les simulations confirment qu'il est de plus en plus délicat de détecter les observations ambiguës quand les actions stochastiques rajoutent du bruit, et donc des variations dans les fonctions Q. Si l'observation E reste assez facile à détecter, ce n'est pas le cas de B. Cependant, dans chaque expérience, à part pour $q = 0.25$, notre algorithme donne de meilleures performances que le Q-Learning classique.

4.2. *Environnement dynamique*

Nous avons aussi testé notre algorithme sur un environnement dynamique, plus complexe. Ici, le monde est plus grand, et un chat doit apprendre à attraper une souris. La figure 5 donne une représentation de cet environnement. Le chat a le choix entre trois actions : *TournerGauche*, *TournerDroite* et *Avancer*. L'état est donné par les positions du chat et de la souris sur la grille, soit $49^2 = 2401$ états. Cependant le chat ne voit que quelques cases autour de lui : deux devant, une à droite, une à gauche. Chacune de ces cases peut être soit *Vide* soit occupée par un *Mur* ou la *Souris*. Il y a donc $3 \times 3 \times 3 = 81$ observations différentes. La récompense reçue par le chat est de $-1,0$ s'il percute un mur et de $+1,0$ si la souris est dans la case directement devant lui.

La souris n'apprend pas, elle a un *comportement programmé et automatique*. La souris n'est pas statique mais ne bouge que quand le chat est sur le point de l'attraper, c'est-à-dire quand la souris est dans la case immédiatement devant lui. Les mouvements de la souris déterminent la difficulté de la tâche et nous avons utilisé deux stratégies de mouvement différentes pour la souris.

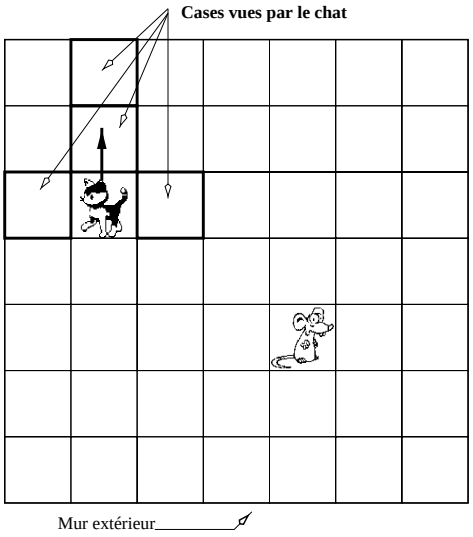


Figure 5. *Chat et souris. Le chat, dont le champ de vision est limité, doit attraper la souris. Mais cette dernière bouge quand elle sur le point de se faire attraper. Suivant la stratégie de déplacement de la souris, qui est fixée, nous pouvons avoir des POMPD qui n'admettent pas d'observable exhaustif fini.*

4.2.1. *Souris fainéante*

La première stratégie de mouvement de la souris est assez simple. Soit elle avance de trois cases en s'éloignant du chat soit, avec la même probabilité, elle se déplace de deux cases vers la droite du chat. Dans le cas ou la souris se trouve bloquée par un mur, elle se place alors juste derrière le chat. Dans tous les cas, la souris disparaît du champ de vision du chat. La figure 6 explicite ces mouvements au cas (a).

Pour que le chat puisse suivre la souris qu'il vient de voir, la politique optimale est d'avancer d'une case puis de tourner sur sa droite pour vérifier si la souris y est. Si elle y est, il doit alors avancer pour l'attraper. Sinon, il doit se tourner vers sa gauche et de nouveau avancer jusqu'à ce qu'il trouve la souris. Evidemment, c'est un peu plus compliqué car la souris peut aussi se trouver derrière le chat. Sans ce cas particulier qui est moins fréquent, cette partie de politique optimale peut s'apprendre avec des trajectoires d'ordre deux. Ce n'est qu'une partie de la politique optimale car le chat doit aussi retrouver la souris quand il n'a aucune idée de là où elle se trouve. Dans ce dernier cas, il n'y a aucune politique d'ordre fini qui permette de trouver le plus court chemin entre le chat et la souris. Nous sommes à la limite du champ d'application théorique de notre algorithme. Nos résultats confirment cette analyse sommaire du

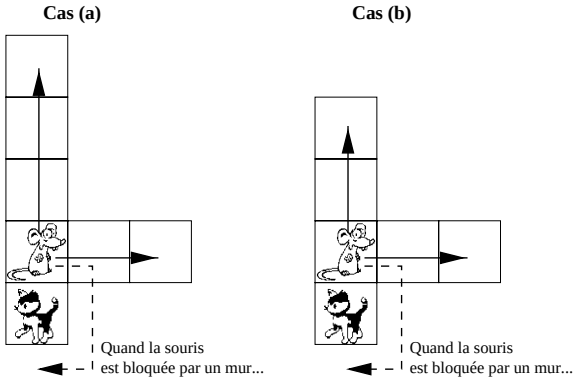


Figure 6. *Stratégie de mouvement de la souris. Avec une probabilité uniforme, elle s'éloigne de trois (deux dans le cas (b)) cases vers l'avant ou sur la droite du chat. Si jamais la souris rencontre un mur, elle vient se placer derrière le chat. Le cas (b) est utilisé quand la souris peut aussi se téléporter.*

problème. Le chat trouve une politique quasi optimale (elle n'est optimale qu'une fois que le chat a vu la souris et qu'il ne la perd pas). Mais, alors que nous pensions qu'une telle politique pouvait être trouvée avec un observable d'ordre deux, notre algorithme utilise un observable d'ordre trois. Cela peut s'expliquer par le fait qu'un observable d'ordre trois est plus robuste qu'un observable d'ordre deux. De plus, comme nous l'avons vu, le chat peut parfois se trouver confronté à des cas particuliers comme quand la souris se place derrière lui ou qu'il la perd... La figure 7 montre en tous cas que les performances sont meilleures quand le chat utilise un observable d'ordre trois.

4.2.2. *Souris qui se téléporte*

Cette fois la souris a des mouvements plus simples, comme le montre le cas (b) de la figure 6. Dans ce cas, une politique markovienne devrait suffire à suivre la souris. Mais la souris peut aussi, toutes les 3000 itérations, se téléporter sur n'importe quelle autre case du monde. Dès lors, il n'existe aucune politique optimale adaptée d'ordre fini, et la politique markovienne n'est plus que quasi optimale.

Si nous utilisons un algorithme de Q-Learning classique, comme le montre la figure 8, le chat a beaucoup de difficulté à retrouver la souris une fois qu'il l'a perdue. Il doit en effet oublier sa politique quasi optimale qui le faisait tourner sur sa droite quand la souris n'était plus en vue. Le chat doit maintenant explorer le monde pour retrouver la souris alors que sa politique quasi optimale le faisait rester sur place.

Avec notre algorithme, le chat apprend une politique d'ordre 2 qui lui permet de passer facilement de la phase où il doit chasser la souris à la phase où il doit la chercher dans le monde. C'est comme si le chat avait appris deux politiques ainsi que leur contexte d'application. La figure 9 montre que les performances en sont améliorées.

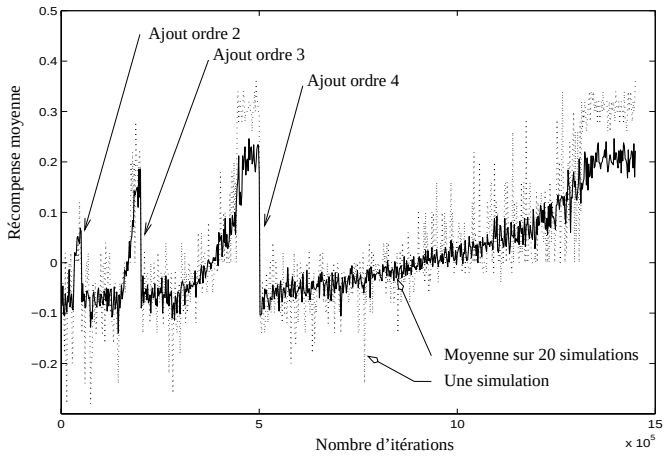


Figure 7. *La souris se déplace doucement. Alors qu'un observable d'ordre deux semblait suffire à trouver une politique quasi optimale, l'agent apprenant préfère utiliser un observable d'ordre 3 pour être plus robuste et couvrir plus de cas d'ambiguïté*

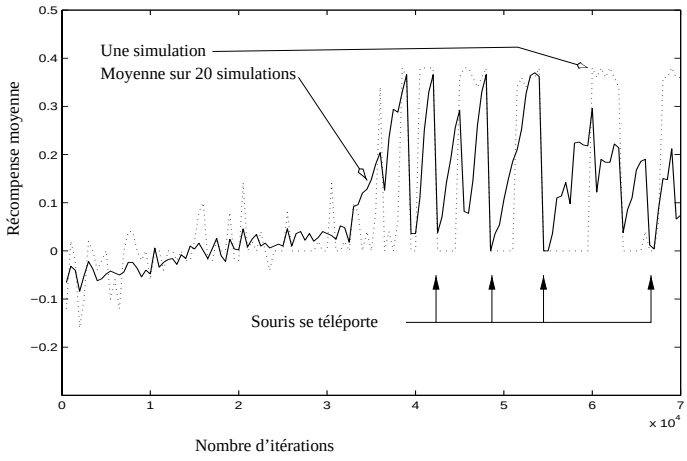


Figure 8. *Q-Learning classique. Une politique markovienne ne suffit pas quand la souris disparaît soudainement de la vue du chat qui doit alors ré-apprendre à explorer le monde.*

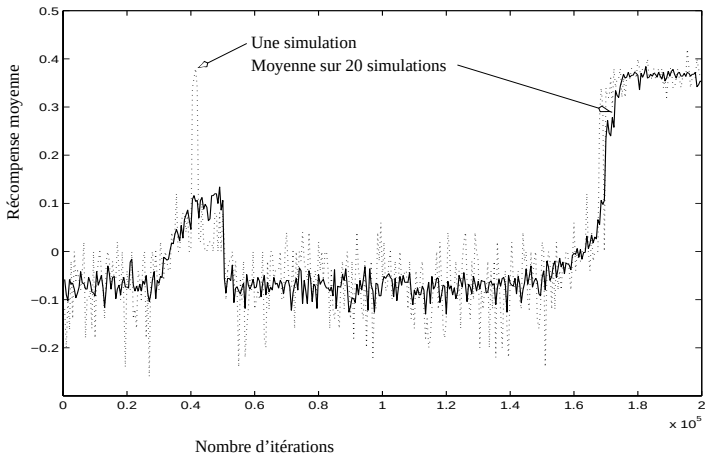


Figure 9. *Notre algorithme. En utilisant des observables d'ordre deux, le chat apprend à changer de politique pour chasser la souris et explorer son environnement. Nous ne voyons plus les baisses de performance quand la souris disparaît.*

La politique d'ordre deux apprise par le chat pour explorer son environnement est illustrée sur la figure 10. On y voit que le chat essaie de suivre les murs extérieurs du monde pour maximiser les cases qu'il couvrira dans son champ de vision durant ses déplacements. En combinant cette tactique avec des actions aléatoires qui peuvent toujours survenir grâce au caractère stochastique des politiques du Q-Learning de Boltzmann, c'est en fait une politique très efficace. Il est vrai que des politiques d'ordre plus élevé seraient plus performantes, mais l'accroissement effectif de performance est si petit que notre algorithme s'est contenté d'un observable d'ordre deux.

Ainsi, notre algorithme n'a pas pu trouver de politique optimale car il n'en existe aucune qui soit d'ordre fini. Mais il a trouvé rapidement une politique d'ordre deux qui donne de très bons résultats.

5. Travaux et recherches similaires

Notre algorithme s'inspire d'idées issues d'autres algorithmes comme, par exemple, le *G-Algorithm* de [CHA 91], le *Q-Learning adapté* de [JAA 94b], l'utilisation de réseaux de neurones pour l'apprentissage par renforcement comme dans [LIN 92] et des théories avancées par [AST 65] et [SON 71]. Son principe est très similaire aux travaux présentés par [RIN 94] et par [MCC 96] et reste assez proche des techniques mises en œuvre dans les modèles de Markov cachés (HMM) [LEV 83].

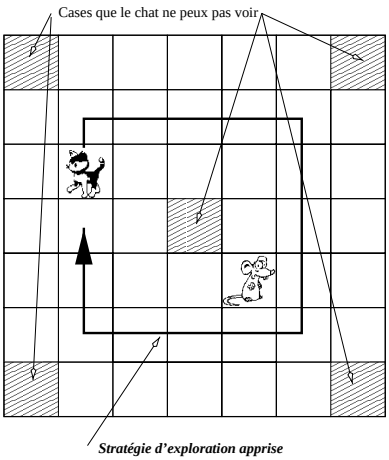


Figure 10. *Explorer l'environnement. C'est en suivant les murs extérieurs que le chat a le plus de chance de retrouver la souris qui n'a plus que 5 cases pour se cacher. Comme la politique reste légèrement aléatoire, le chat retrouve la souris en très peu de temps.*

– **POMDP et "belief states".** Dans ses travaux sur les POMDP, Aström utilise une extension d'état qui s'appuie sur des distributions de probabilité sur les états du MDP sous-jacent. Ces états étendus sont appelés *belief states*. Cette extension d'état permet de traiter le problème comme un MDP mais on doit alors raisonner sur un espace d'état qui est continu et donc potentiellement infini. Sondik [SON 71] montre cependant que l'utilité sur cet espace d'état peut être approchée par une fonction continue, théorie utilisée par Littman [LIT 94] puis Cassandra [CAS 98] qui proposent des algorithmes pour résoudre les POMDP, dont le fameux algorithme *WITNESS*. Malgré diverses améliorations, cette famille d'algorithmes ne peut que difficilement traiter des problèmes où il y a plus d'une dizaine d'états sous-jacents. Il apparaît en fait que l'extension d'état proposée par Aström est un résumé exhaustif de tout le passé du processus. Nous utilisons aussi une extension d'état mais de taille plus réduite car nous n'étendons que les observations ambiguës. Le coût de la recherche de cette extension appropriée est compensé par la facilité qu'il y a ensuite à trouver une politique optimale. Nous pouvons donc aborder des problèmes d'une taille plus grande. De plus, les algorithmes qui s'appuient sur les *belief states* ont besoin de connaître le modèle d'évolution du processus, alors que nous n'en avons pas besoin.

– **Q-Learning adapté.** En s'appuyant sur les remarques et les limitations du Q-Learning classique utilisé dans les POMDP, Jaakkola *et al.* [JAA 94b] propose un algorithme de renforcement qui construit des politiques markoviennes stochastiques. Bien que les politiques ainsi définies soient sous-optimales, ces travaux, avec ceux de Singh *et al.* [SIN 94], sont parmi les premiers à étudier le comportement des algorithmes classiques quand les observations sont partielles. Nous avons largement profité

des remarques pertinentes et de la compréhension gagnée à travers ces travaux pour définir nos heuristiques de recherche de trajectoires d'observation-action ambiguës.

– **Q-Learning récurrent.** L'idée principale des travaux de Lin [LIN 92] est d'utiliser un réseau de neurones pour calculer une approximation de la fonction d'utilité du processus. Pour utiliser l'information contenue dans le passé du processus, il utilise des réseaux récurrents comme ceux proposés par Elman [ELM 90] ou Williams et Zipser [WIL 90]. Ces architectures lui permettent de stocker dans les couches internes de ses réseaux des informations pertinentes sur le passé du processus, information qui est ensuite utilisée dans une architecture d'apprentissage par renforcement neuronale plus classique. Les résultats obtenus sont intéressants mais visiblement sous-optimaux. La raison majeure est que la formation des états internes de l'architecture neuronale n'est pas supervisée, l'information qui y est traitée n'est que peu utile pour la résolution du POMDP. Il est difficile de détecter l'information utile dans le passé du processus. Nous avons choisi de diriger plus précisément cette recherche d'éléments pertinents dans notre algorithme en nous concentrant sur l'ambiguïté des observations. Si nous pouvons alors construire une extension d'état plus efficace, nous n'avons pas la puissance d'expression et de généralisation que l'utilisation de réseaux neuronaux permet.

– **G-Algorithme.** Cet algorithme proposé par Chapman [CHA 91] apprend à distinguer les observations ambiguës des observations non ambiguës. L'un des prérequis de cet algorithme est que l'état sous-jacent actuel du processus soit accessible par le biais d'une ou de plusieurs actions de perception. Le but ici n'est pas de trouver une politique optimale quand les informations sont partielles mais d'utiliser les bonnes informations parmi toutes celles qui sont disponibles dans un état donné. L'algorithme construit un arbre de décision pour choisir parmi les différents éléments des observations. Le principal intérêt de cet algorithme est l'utilisation d'un test statistique pour différencier les distributions de probabilité des récompenses reçues à chaque nœud de l'arbre. Tel quel, l'algorithme a besoin de quelques adaptations pour être utilisé dans le contexte des POMDP d'ordre N.

– **U-Tree.** L'algorithme de McCallum [MCC 96] s'appuie lui aussi fortement sur la structure arborescente proposée par Chapman (G-Algorithm) et Ron [RON 94]. Chaque feuille de l'arbre correspond à une trajectoire d'observation non-ambiguë. L'arbre est construit itérativement. McCallum détecte les noeuds ambigus par un test statistique sur les distributions de probabilité de récompense entre les feuilles de l'arbre et une extension virtuelle de cet arbre qu'il appelle la *frange*. Une différence de distribution indique une feuille ambiguë et l'arbre est alors étendu et on lui rajoute une nouvelle frange. De plus, comme pour l'algorithme de Chapman, il peut différencier les observations en n'utilisant que quelques caractéristiques de ces dernières pour les regrouper en différentes classes. Ainsi, l'algorithme reste performant même avec des espaces d'états et d'observations conséquents.

Le désavantage de cette méthode est qu'elle est gourmande en termes de calcul et de mémoire. Pour pouvoir utiliser des tests statistiques, il faut stocker pour chaque nœud tout son historique en termes de récompense. Chaque itération de l'algorithme est assez coûteuse car il faut mettre à jour plusieurs nœuds de la frange. Le test statistique est lui-même gourmand en ressource et en temps de calcul. Enfin, bien que nous pensons que notre algorithme et celui de McCallum s'appuient sur les mêmes

propriétés mathématiques, aucune étude théorique n'est envisagée par McCallum qui se contente de quelques suppositions.

Par contre, l'algorithme a été testé avec succès sur plusieurs problèmes, dont le *Cheese Maze*. McCallum teste aussi son algorithme sur un problème complexe de son invention : un agent doit apprendre à conduire dans un environnement simulé composé de 4 voies d'autoroute où il faut éviter des camions et des voitures. Pour cet environnement composé de plus de 3500 états et 2500 observations, l'algorithme du U-Tree est capable de proposer une politique qui, si elle n'est pas optimale, donne de meilleurs résultats que toutes les politiques construites "à la main" par McCallum.

Nous venons de le rappeler, l'algorithme mis au point par McCallum n'est pas assuré de trouver une politique optimale. En fait, cette question de l'optimalité n'est que peu abordée par l'auteur, ce qui est lié au fait qu'il manque une étude théorique de son algorithme. Néanmoins, McCallum pense que son algorithme peut faire face à la fois au bruit *structurel* (observation ambiguës) et au bruit *perceptuel* (fonction d'observation bruitée). Bien que l'étude théorique de notre algorithme (voir [DUT 99]) se soit limitée au cas de fonctions d'observation déterministes, nous pensons que notre algorithme devrait converger en utilisant une large gamme de fonctions d'observation stochastiques, nous en reparlons à la section 6.

– **Modèles de Markov cachés.** Des méthodes bien connues initialement proposées par Levinson *et al.* [LEV 83] permettent de trouver la structure des modèles de Markov Cachés répondant au mieux à un problème posé. Ces modèles ne prennent pas en compte l'aspect *décision* des POMDPs mais ils sont capables de gérer le bruit d'observation. Cependant, même s'il paraît assez facile de modifier ces algorithmes pour prendre en compte les actions, nous ne pensons pas que nous pourrions utiliser ces algorithmes modifiés pour notre classe de problème car les méthodes de Levinson ne peuvent qu'adapter les paramètres d'un modèle où le nombre de nœuds cachés est connu à l'avance. Or, ce n'est clairement pas notre cas. Il serait possible, comme il est fait généralement, de tester plusieurs structures et de choisir la plus vraisemblable, mais cette option paraît délicate à mettre en œuvre à cause de son coût en ressource de calcul. Par contre, de nouvelles approches pour les modèles de Markov cachés essayent de construire incrémentalement la structure du modèle, comme par exemple les travaux de Stolcke et Omohundro[STO 92]. Malgré tout, prendre en compte le bruit de perception avec ces méthodes appliquées aux POMDP risque d'être beaucoup plus complexe que dans le cas des modèles de Markov cachés.

6. Améliorations et discussions

Pour être plus efficace et pour pouvoir être utilisé sur des problèmes réels, notre algorithme nécessite encore quelques améliorations. Nous en détaillons quelques-unes ci-dessous.

– **Détection des états ambigus.** Nous utilisons des heuristiques pour décider si un état étendu est ambigu, en nous appuyant sur le fait que la fonction Q converge plus lentement pour ces états ambigus. Nous travaillons donc principalement avec les

variations de la fonction Q, mais de nombreux facteurs influencent ces variations, comme par exemple la fonction de récompense qui est stochastique, les variations de la fonction Q dans les états voisins, les probabilités de transition. Le vrai problème est de déterminer la source des variations pour savoir si elle provient d'une observation partielle ou de l'aspect aléatoire de l'environnement. L'algorithme du *U-Tree* de [MCC 96] utilise un test statistique plus rigoureux mais il reste confronté au même problème : l'origine de l'aléa. Améliorer la détection de ces ambiguïtés est un point crucial de l'algorithme, mais nous n'avons pas pour l'instant de piste de recherche prometteuses à ce sujet.

– **Influence du bruit de perception.** Notre étude théorique ne s'est faite que dans le cas d'une fonction d'observation f déterministe (en particulier, pour pouvoir parler de classe d'équivalence). Intuitivement, notre algorithme devrait pouvoir converger avec certaines fonctions d'observations stochastiques. Par exemple, des fonctions injectives, c'est-à-dire quand une observation n'est liée qu'à un état, ne devraient poser aucun problème. Par contre, quand une observation peut être l'image de deux états différents par une fonction stochastique, la difficulté augmente. En fait, cela revient à ajouter une cause de variation supplémentaire pour la fonction Q. Notre algorithme devrait toujours pouvoir converger, mais avec plus de difficultés. Dans la plupart des cas, on ne devrait plus se situer dans la classe des POMDPs d'ordre fini, ce qui n'empêche pas forcément notre algorithme de bien se comporter.

– **Observations pertinentes séparées par un grand intervalle de temps.** Notre algorithme n'utilise que des observables formés d'événements passés contigus. Dès lors, si ce qui est important pour diminuer l'ambiguïté du présent se situe il y a 50 pas de temps, il faudrait des trajectoires d'ordre 50, ce qui n'est pas envisageable pour l'instant. Une idée, comme celle mise en oeuvre par McCallum, est d'étiqueter les divers éléments d'une trajectoire en indiquant le nombre d'itérations dont il faut remonter dans le passé pour prendre en compte cette observation. Reste alors à mettre à jour cette étiquette, ce qui en pratique ne peut se faire qu'en testant toutes les valeurs de l'étiquette, ce qui devient rapidement infaisable. Nous travaillons actuellement sur ce problème lui aussi très important en essayant, par le biais d'une mesure d'information ou de pertinence, de détecter les observations qui réduisent l'ambiguïté et en ne travaillant plus avec des trajectoires "continues" mais "discontinues".

De plus, notre algorithme souffre de limitations qui sont plus générales et qui sont communes à la plupart des algorithmes d'apprentissage par renforcement. Ces limitations sont actuellement au centre de plusieurs recherches sur l'apprentissage par renforcement.

– **Explosion combinatoire.** La complexité de notre algorithme est plus grande que celle des algorithmes classiques comme le Q-Learning ou TD(λ), nous sommes encore plus dépendant du nombre d'actions, d'états et d'observations. Des solutions maintenant classiques comme par exemple des estimations paramétriques de fonctions, des réseaux de neurones ou des codages appropriés pourraient êtres utilisées pour ne pas avoir à associer à chaque élément une utilité mais pour se contenter d'estimer l'utilité sur les différents observables.

– **Vitesse de convergence.** En théorie, il faut que chaque élément d'un observable soit visité une infinité de fois pour être sûr de la convergence du Q-Learning, en admettant que l'observable soit exhaustif. Ici, c'est l'exploration de l'environnement qui permet d'améliorer cette vitesse de convergence. A la place d'une exploration aléatoire de l'environnement, nous pourrions utiliser une exploration dirigée. En plus des méthodes décrites par [SUT 91], [THR 92] ou [MOO 93] nous pouvons aussi accorder plus d'importance aux nouveaux éléments rajoutés à l'observable qu'aux anciens éléments qui sont non-ambigus et pour lesquels une action jugée optimale existe déjà.

– **Algorithme réactif.** Notre algorithme, ainsi que les algorithmes d'apprentissage par renforcement en général, n'est pas très réactif. Tel quel, il n'est donc pas très adapté comme système de prise de décision pour des agents autonomes qui doivent réagir en temps réel à des changements ou des modifications de leur environnement. Même si l'environnement n'évolue pas, il faut disposer d'un bon simulateur sur lequel faire des expériences avant d'utiliser la politique apprise sur un robot, car il faut souvent des milliers d'itérations, parfois destructrices, pour trouver cette politique. Il n'est donc pas envisageable de la faire apprendre directement par le robot. Des idées inspirées par les méthodes mises en œuvre dans [STO 92] peuvent éventuellement permettre de rendre les algorithmes plus réactifs.

7. Conclusion

Dans cet article, nous avons étudié l'utilisation de techniques d'apprentissage par renforcement pour les processus de décision markoviens partiellement observés. Dans ce cadre, nous avons proposé un nouvel algorithme qui s'appuie sur la construction incrémentale d'un observable exhaustif minimal. L'extension de l'espace d'états ainsi définie permet de se ramener à un processus de décision qui est alors markovien dont on peut chercher une solution optimale. Cette politique optimale est en même temps une politique adaptée optimale pour le processus markovien partiellement adapté.

Nous avons donné le cadre théorique d'application de notre algorithme, c'est-à-dire l'ensemble des POMDP d'ordre fini. Nous avons ensuite validé notre algorithme par des simulations sur des problèmes de difficulté croissante. Ces simulations nous ont donné une meilleure compréhension du comportement de notre algorithme. Par ailleurs, elles ont montré la robustesse de notre algorithme qui reste performant pour des problèmes se situant hors de son cadre théorique de convergence.

En comparant notre travail avec d'autres travaux sur les processus de décisions markoviens partiellement observés nous avons vu que notre algorithme pouvait trouver une solution optimale à des problèmes où l'algorithme de Jaakkola était limité à des solutions sous-optimales. De plus notre algorithme peut être utilisé quand le modèle d'évolution du processus n'est pas connu, ce qui n'est pas le cas de tout les algorithmes qui s'appuient sur les *belief states*, comme WITNESS par exemple. En fait, notre algorithme est très proche de l'algorithme du *U-Tree* proposé par McCallum qui part des mêmes idées, sans toutefois s'appuyer sur une formalisation mathématique rigoureuse. Néanmoins, le *U-Tree* détecte les observations ambiguës au moyen d'un

test statistique ce qui est à la fois plus adéquat, mais aussi plus lourd en termes de temps et de ressource de calcul.

En plus des limitations générales de tous les algorithmes d'apprentissage par renforcement (explosion combinatoire, convergence lente, faible réactivité...) notre algorithme se trouve confronté à des problèmes spécifique dus aux observations. Si la fonction d'observation est trop bruitée, notre algorithme peut ne pas converger et, s'il converge, la solution trouvée ne sera pas forcément optimale. Notre algorithme se repose sur des heuristiques pour détecter les éléments ambigus d'un observable, il serait intéressant d'utiliser de meilleures heuristiques pour ces détections. En fait, le plus gros problème auquel est confronté notre algorithme est celui de distinguer les différentes sources d'aléa sur les observations : bruit structurel (observations ambiguës), bruit du modèle (fonction de transition et de récompense) et bruit d'observation (fonction d'observation bruitée). Ce problème est encore largement ouvert.

Enfin, pour que notre algorithme puisse être utilisé pour des agents autonomes, il manque de réactivité. Des travaux dans cette direction sont en cours, mais là encore les difficultés à résoudre sont importantes.

8. Bibliographie

[AST 65] ASTRÖM K., « Optimal control of markov decision processes with incomplete state estimation », *Journal of Mathematical Analysis and Applications*, vol. 10, 1965, p. 174–205.

[BEL 57] BELLMAN R., *Dynamic programming.*, Princeton University Press, Princeton, New-Jersey., 1957.

[CAS 98] CASSANDRA A., « Exact and Approximate Algorithms for Partially Observable Markov Decision Processes », PhD thesis, Brown University, Department of Computer Science, Providence, RI, 1998.

[CHA 91] CHAPMAN D., KAELBLING L., « Input generalization in delayed reinforcement learning : an algorithm and performance comparisons. », *International Joint Conference in Artificial Intelligence, Sydney.*, 1991.

[DUT 99] DUTECH A., « Apprentissage d'environnements : approches cognitives et comportementales », PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, France, 1999.

[ELM 90] ELMAN J., « Finding structure in time. », *Cognitive Science*, vol. 14, 1990, p. 179–211.

[JAA 94a] JAAKKOLA T., SINGH S., JORDAN. M., « On the convergence of stochastic iterative dynamic programming algorithms. », *Neural Computation*, vol. 6, 1994, p. 1186–1201.

[JAA 94b] JAAKKOLA T., SINGH S., JORDAN M., « Reinforcement learning algorithm for partially observable Markov decision problems. », TESAURO G., TOURETSKY D., LEEN T., Eds., *Advances in neural information processing systems*, vol. 7, MIT Press, Cambridge, Massachusetts, 1994.

[LEV 83] LEVINSON B., RABINER L., SONDHI. M., « An introduciton to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. », *The Bell System Technical Journal*, vol. 62, n° 4, 1983.

[LIN 92] LIN. L.-J., « Self-improving reactive agent based on reinforcement learning, planing and teaching. », *Machine Learning*, vol. 8, 1992, p. 293–321.

[LIT 94] LITTMAN M., « The WITNESS algorithm : solving partially observed markov decision processes », rapport n° CS–94–40, 1994, Computer Science Dept, Brown University, Providence, Rhode Island.

[MCC 95] MCCALLUM A., « Reinforcment learning with selective perception and hidden state », PhD thesis, Dept. of Computer Science, University of Rochester, Rochester, New York., 1995.

[MCC 96] MCCALLUM A., « Learning to use selective attention and short-term memory in sequential tasks », *Proceedings of the Fourth International Conference on Simulating Adaptive Behavior*, MIT Press, Cambridge, Massachusetts, 1996.

[MOO 93] MOORE A., ATKESON C., « Memory based reinforcement efficient computation by prioritized sweeping. », HANSON S., GILES C., COWAN J., Eds., *Advances in neural information processing systems*, vol. 5, Morgan Kaufmann, 1993.

[PUT 94] PUTERMAN M., *Markov Decision Processes : discrete stochastic dynamic programming*, John Wiley & Sons, Inc. New Yrok, NY, 1994.

[RIN 94] RING M., « Continual learning in reinforcement environments. », PhD thesis, University of Texas, Austin, Texas, 1994.

[RON 94] RON D., SINGER Y., TISHBY N., « Learning probabilistic automata with variable memory length. », *Proceedings Computational Learning Theory, ACM Press.*, 1994.

[SIN 94] SINGH S., JAAKKOLA T., JORDAN M., « Learning without state estimation in partially observable markovian decision processes. », *Proceedings of the Eleventh International Conference on Machine Learning.*, 1994.

[SON 71] SONDIK E., « The optimal control of partially observable markov decision processes. », PhD thesis, Stanford University, California, 1971.

[STO 92] STOLCKE A., OMOHUNDRO S., « Hidden Markov model induction by bayesian model merging. », *Advances in neural information processing systems*, vol. 4, 1992.

[SUT 88] SUTTON R., « Learning to predict by the methods of temporal differences », *Machine Learning*, vol. 3, 1988, p. 9–44.

[SUT 91] SUTTON R., « Integrated modeling and control based on reinforcement learning and dynamic programming », MOODY J., LIPPMAN R., TOURETSKY D., Eds., *Advances in neural information processing systems*, vol. 3, Morgan Kaufmann, 1991.

[THR 92] THRUN S., « Efficient exploration in reinforcement learning. », rapport n° CMU-CS-92-102, 1992, Computer Science Department, Carnegie Mellon University.

[WAT 89] WATKINS C., « Learning from delayed rewards. », PhD thesis, King's College of Cambridge, UK., 1989.

[WIL 90] WILLIAMS R., ZIPSER D., « Gradient-based learning algorithms for recurrent connectionnist networks. », rapport n° NU-CCS-90, 1990, Northeast University, College of Computer Science, Boston.

Appendix A : algorithmes de Q-Learning classiques

Le Q-Learning est un algorithme classique d'apprentissage par renforcement qui a été proposé par [WAT 89] pour résoudre les MDP quand le modèle d'évolution est inconnu. Cet algorithme s'appuie sur l'équation de point fixe de [BEL 57] qui dit que l'utilité pondérée en horizon infini V_{p^*} pour la politique optimale p^* est telle que :

$$V_{p^*}(s) = max_{a \in A}\{E_{r \times \pi(s,a)}[r(s,a) + \gamma V_{p^*}(\cdot)]\} \quad [1]$$

Le principe du Q-Learning est d'estimer une fonction Q définie par :

$$Q^*(s,a) = E_{r \times \pi(s,a)}[r(s,a) + \gamma V_{p^*}(\cdot)] \quad [2]$$

en utilisant une mise à jour itérative asynchrone donnée par :

$$Q_{n+1}(s_t,a_t) = (1-\alpha_n)Q_n(s_t,a_t) + \alpha_n[r_t + \gamma \cdot max_{a \in A}(Q_n(s_{t+1},a))] \quad [3]$$

où r_t est le renforcement reçu en ayant choisi l'action a_t dans l'état s_t, ce qui a mis le processus dans le nouvel état s_{t+1}. α_n est un réel positif compris entre 0 et 1 que l'on appelle le taux d'apprentissage.

Il a été prouvé, par exemple dans [JAA 94b], que si les espaces d'états et d'actions $\mathcal{S}$ et $\mathcal{A}$ sont finis, si α_n est tel que $\sum_n \alpha_n = \infty$ et que $\sum_n \alpha_n^2 \leq \infty$ et si le processus est récurrent (c'est-à-dire que l'on passe par chaque état-action un nombre infini de fois), cet algorithme converge vers Q^*.

En principe, il faut d'abord explorer aléatoirement l'environnement pendant un grand nombre d'itérations pour que le Q-Learning puisse converger vers la fonction Q optimale et seulement ensuite utiliser la politique optimale définie par : $p^*(s) = argmax_{a \in A}Q^*(s,a)$. L'exploration de l'environnement peut se faire en choisissant aléatoirement les actions à effectuer suivant une distribution de probabilité uniforme comme $P\{a_t = a\} = \frac{1}{|A|}$. C'est ce que nous appelons *Q-Learning Uniforme*.

Une autre façon classique de faire est d'utiliser progressivement la connaissances apprise sur l'environnement et la tâche à accomplir pour décider de la prochaine action. Pour ce faire, on peut utiliser une distribution de probabilité de Boltzman sur les actions de manière à ce que :

$$P\{a_t = a | s_t = s\} = e^{\frac{Q_n(s,a)}{T_t}} / \sum_{b \in A} e^{\frac{Q_n(s,b)}{T_t}} \quad [4]$$

où T_t est un paramètre de température dont la valeur initiale est T_{start} et qui décroit exponentiellement par $T_{t+1} = T_{dec} \cdot T_t$. Ainsi, l'algorithme passe progressivement d'une exploration aléatoire de l'environnement à la politique optimale apprise. Nous appelons cet algorithme le *Q-Learning de Boltzman*.

Appendix B : démonstration des différents théorèmes

Démonstration du Théorème 1. Pour démontrer le Théorème 1 nous allons montrer que les probabilités de transition étendues $\hat{T}$ au temps t dépendent seulement de l'état étendu présent et de l'action choisie. Nous noterons s un élément de $\mathcal{S}$ et $\hat{s}$ un élément de $\hat{\mathcal{S}}$. Σ_t est la trajectoire d'état-action du processus.

Calculons les probabilités de transition quand tout le passé du processus est connu.

$$P\{proj_F[f(\Sigma_{t+1})] = \hat{s}'|A_t = a, \Sigma_t = \sigma\} \quad [5]$$

$$= P\{proj_F[(O_{t+1}, a, f(\Sigma_t))] = \hat{s}'|A_t = a, \Sigma_t = \sigma\} \quad [6]$$

$$= P\{O_{t+1} = o|A_t = a, \Sigma_t = \sigma\} \quad [7]$$

$$= \sum_{s \in f^{-1}(o)} P\{S_{t+1} = s|A_t = a, \Sigma_t = \sigma\} \quad [8]$$

mais nous savons que est un observable exhaustif, donc

$$\sum_{s \in f^{-1}(o)} P\{S_{t+1} = s|a, \Sigma_t\} \quad [9]$$

$$= \sum_{s \in f^{-1}(o)} P\{S_{t+1} = s|A_t = a, \hat{S}_t = proj_F[f(\Sigma_t)]\} \quad [10]$$

$$= \sum_{s \in f^{-1}(o)} P\{S_{t+1} = s|A_t = a, \hat{S}_t = \hat{s}_t\} \quad [11]$$

Ce qui termine la démonstration. □

Pour démontrer le Théorème 2 nous avons besoin du lemme suivant.

Lemme 1. *Soient s et s' deux éléments de $\mathcal{S}$ tels que $r(s,a) = r(s',a)$ et $\pi(s,a) = \pi(s',a)$ pour tout a de $\mathcal{A}$, alors il est indifférent de choisir $p^*(s)$ ou $p^*(s')$ quand on se trouve dans l'état s. Cela signifie que nous avons :*

$$Q_{p^*}(s, p^*(s)) = Q_{p^*}(s, p^*(s')) = V_{p^*}(s) \quad [12]$$

où $V_{p^*}(s)$ est l'utilité optimale de l'état s.

Démonstration du Lemme. En utilisant l'équation de Bellman (cf Equation [1]), nous avons

$$Q_{p^*}(s,a) = r(s,a) + E_{\pi(s,a)}[\gamma V_{p^*}(.)] \quad [13]$$

D'après les propriétés de r et p^* on tire :

$$Q_{p^*}(s',a) = r(s,a) + E_{\pi(s,a)}[\gamma V_{p^*}(.)] = Q_{p^*}(s,a) \quad [14]$$

Par définition d'une politique optimale, on a : $p^*(s) = argmax_{a \in A}\{Q_{p^*}(s,a)\}$ ce qui nous permet d'écrire que, pour tout a : $Q_{p^*}(s,p^*(s)) \geq Q_{p^*}(s,a)$. En particulier, pour $a = p^*(s')$ nous avons

$$Q_{p^*}(s,p^*(s)) \geq Q_{p^*}(s,p^*(s')) = Q_{p^*}(s',p^*(s')) \quad [15]$$

L'égalité se déduit de l'Equation [14].
Comme s et s' ont un rôle symétrique, nous pouvons écrire que :
$Q_{p^*}(s,p^*(s)) = Q_{p^*}(s',p^*(s'))$ d'où l'on déduit que $Q_{p^*}(s,p^*(s)) = Q_{p^*}(s,p^*(s'))$
et, par définition, nous avons : $V_{p^*}(s) = Q_{p^*}(s,p^*(s))$ □

Démonstration du Théorème 2. Nous noterons p une politique de λ et $\hat{p}$ une politique de $\hat{\lambda}$. $V_p(s)$ est l'utilité de l'état s en utilisant la politique p et $\hat{V}_{\hat{p}}(\hat{s})$ est l'utilité de $\hat{p}$. A partir de maintenant, nous considérons les processus à l'instant t, où t est plus grand que n, l'ordre de l'observable exhaustif de λ. Nous allons d'abord démontrer que $\hat{V}_{\hat{p}^*} \leq V_{p^*}$, avant de montrer que $V_{p^*} \leq \hat{V}_{\hat{p}^*}$

Soit p' la politique définie par

$$p'(\Sigma_t) = \hat{p}^*(proj_F[f(\Sigma_t)]) = \hat{p}^*(\hat{s}_t) \quad [16]$$

Même si p' est stochastique, son utilité ne peut être supérieure à celle de p^* car λ étant un processus décisionnel markovien, il admet une politique *déterministe* optimale (ici p^*). Ainsi nous avons

$$V_{p'}(\Sigma_t) \leq V_{p^*}(s_t) \quad [17]$$

De plus, par définition de p', nous savons que :

$$V_{p'}(\Sigma_t) = \hat{V}_{\hat{p}^*}(\hat{s}_t) \quad [18]$$

Nous démontrons ainsi que

$$\hat{V}_{\hat{p}^*} \leq V_{p^*} \quad [19]$$

Soient deux états s et s' équivalents pour $\hat{\lambda}$. Par définition de l'équivalence, on a bien :

$$\forall a \in \mathcal{A}, r(s,a) = r(s',a) \quad [20]$$

De plus, comme $\hat{\lambda}$ est défini à partir d'un observable exhaustif, on obtient pour tout a de $\mathcal{A}$ et s'' de $\mathcal{S}$:

$$P\{s''|a,\hat{s}\} \quad [21]$$

$$P\{s''|a,\hat{s'}\} \quad [22]$$

s et s' remplissent bien les conditions du lemme 1. On peut alors construire une politique adaptée $\hat{p}'$ par $\hat{p}'(\hat{s}) = p^*(s)$ où $s \in S$ est un élément de la classe d'équivalence définie par $\hat{s}$. Et comme d"après le lemme 1 l'utilité de chacun des éléments de la classe d'équivalence de $\hat{s}$ est identique et vaut $V_{p^*}(s)$, on a bien :

$$\hat{V}_{\hat{p}'}(\hat{s}) = V_{p^*}(s) \quad [23]$$

Par définition de la politique optimale sur $\hat{\lambda}$, on sait que :

$$\hat{V}_{\hat{p}'}(\hat{s}) \leq \hat{V}_{\hat{p}^*}(\hat{s}) \quad [24]$$

c'est-à-dire :

$$V_{p^*}(s) \leq \hat{V}_{\hat{p}^*}(\hat{s}) \quad [25]$$

De équations [19] et [25], on tire que :

$$V_{p^*}(s) = \hat{V}_{\hat{p}^*}(\hat{s}) \quad [26]$$

ce qui termine la démonstration. □

Développement autonome des comportements de base d'un agent

Olivier Buffet[*,**] — **Alain Dutech**[**] — **François Charpillet**[**]

[*] *National ICT Australia / The Australian National University*
RSISE Building 115 - ANU / Canberra ACT 0200 / Australia
olivier.buffet@nicta.com.au

[**] *Loria - INRIA-Lorraine / Campus Scientifique - BP 239*
54506 Vandœuvre-lès-Nancy cedex / France
{dutech,charp}@loria.fr

RÉSUMÉ. *La problématique abordée dans cet article est celle de la conception automatique d'agents autonomes devant résoudre des tâches complexes mettant en œuvre plusieurs objectifs potentiellement concurrents. Nous proposons alors une approche modulaire s'appuyant sur les principes de la sélection d'action où les actions recommandées par plusieurs comportements de base sont combinées en une décision globale. Dans ce cadre, notre principale contribution est une méthode pour qu'un agent puisse définir et construire automatiquement les comportements de base dont il a besoin* via *des méthodes d'apprentissage par renforcement incrémentales. Nous obtenons ainsi une architecture très autonome ne nécessitant que peu de réglages. Cette approche est testée et discutée sur un problème représentatif issu du "monde des tuiles".*

ABSTRACT. *The problem addressed in this article is that of automatically designing autonomous agents having to solve complex tasks involving several –and possibly concurrent– objectives. We propose a modular approach based on the principles of action selection where the actions recommanded by several basic behaviors are combined in a global decision. In this framework, our main contribution is a method making an agent able to automatically define and build the basic behaviors it needs through incremental reinforcement learning methods. This way, we obtain a very autonomous architecture requiring very few hand-coding. This approach is tested and discussed on a representative problem taken from the "tile-world".*

MOTS-CLÉS : *problèmes de décision markoviens, apprentissage par renforcement, motivations multiples.*

KEYWORDS: *Markov Decision Problems, Reinforcement Learning, Multiple Motivations.*

RSTI - RIA – 19/2005. Apprentissage automatique, pages 603 à 632

1. Introduction

Un agent (Russell *et al.*, 1995, Jennings *et al.*, 1998) est défini ici comme une entité immergée dans un environnement au sein duquel cette entité cherche à atteindre un objectif qui lui est propre. Dans ce but, l'agent interagit avec son environnement : il le perçoit et peut agir sur celui-ci, comme illustré sur la figure 1.

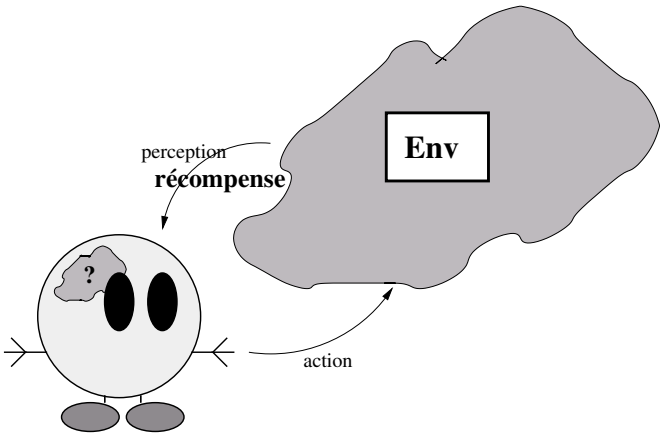


Figure 1. *Agent et son environnement liés à travers une boucle perception/action*

Dans nos travaux, nous cherchons à concevoir de manière automatisée des agents se trouvant dans des milieux à la fois inconnus, incertains et partiellement observés (l'état précis du système agent+environnement n'est pas connu). Sous ces hypothèses difficiles, nous nous limitons à considérer des agents réactifs, adoptant un comportement de type stimulus-action.

Notre intérêt se porte sur la synthèse de ce comportement par l'utilisation de méthodes d'*apprentissage par renforcement* (A/R). Dans ce cadre, l'objectif assigné à un agent est défini par des récompenses lui indiquant quand un événement considéré comme positif ou négatif survient. L'apprentissage par renforcement consiste pour un agent à renforcer les décisions qui lui permettent de maximiser les récompenses qu'il reçoit. Si de nombreuses recherches ont déjà été conduites sur cette thématique, nous nous intéressons ici plus particulièrement à la situation dans laquelle un agent possède plusieurs motivations éventuellement conflictuelles.

L'apprentissage par renforcement passe classiquement par le formalisme des problèmes de décision markoviens (MDP). Si ceux-ci peuvent *a priori* tenir compte de différentes motivations du moment qu'elles se traduisent par des fonctions de récompenses additives ($r = r_1 + r_2 + \dots$), il devient vite déraisonnable d'apprendre un comportement dans de grands espaces d'états. Dans l'hypothèse que l'environnement est constitué d'objets en nombre variable et que les motivations courantes de l'agent dépendent des objets présents, une approche possible est celle des MDP *relationnels* (Boutilier *et al.*, 2001, Dzeroski *et al.*, 2001, Gretton *et al.*, 2004) (voir figure 2). Mal-

heureusement, on ne peut aisément étendre cette approche dans le cas d'un agent avec observations partielles et sans mémoire à court terme (agent à comportement réactif), situation dans laquelle il faut chercher une politique stochastique, donc difficilement représentable par des outils de logique comme ceux utilisés par l'A/R relationnel.

Figure 2. *Les approches relationnelles pour l'apprentissage par renforcement permettent entre autres d'utiliser la même connaissance pour mettre le bloc* `[A]` *sur* `[C]` *dans le cas de gauche que pour mettre* `[C]` *sur* `[B]` *ou* `[D]` *(même couleurs) dans le cas de droite. Dans ce second cas, on a deux motivations concurrentes : mettre* `[C]` *sur* `[B]` *ou mettre* `[C]` *sur* `[D]`

1.1. *Approche : sélection d'action et apprentissage par renforcement*

L'apprentissage par renforcement relationnel classique répondant mal au problème de la prise de décision avec motivations multiples et en milieu partiellement observé (sans mémoire à court terme), nous avons orienté nos recherches vers des méthodes heuristiques de combinaison de comportements (on parle de sélection d'action (Tyrrell, 1993), voir section 2.3) pour voir comment elles peuvent entrer dans la conception d'un agent par apprentissage par renforcement.

Suivant les mêmes principes que (Humphrys, 1996), nous employons une architecture de sélection d'action dans laquelle un agent connaît différents comportements de bas niveau (que nous appelons *comportements de base*) et doit choisir une action d'après l'utilité/la préférence que lui associent ces comportements de base. De précédents travaux dans le domaine de l'A/R pour la sélection d'action se sont concentrés :

– soit sur l'adaptation du processus de combinaison de comportements de base (Buffet *et al.*, 2002),

– soit sur l'apprentissage/l'amélioration de comportements de base sélectionnés au préalables (Humphrys, 1996, Buffet *et al.*, 2003).

Alors que ces travaux proposent de fournir à l'agent des comportements sélectionnés manuellement, notre objectif est que l'agent puisse définir *de lui-même* les comportements de base dont il a besoin pour prendre de bonnes décisions, de manière :

– à compléter, voire remplacer des choix intuitifs de ces comportements, et

– à automatiser autant que possible la conception.

Pour cela, nous suggérons de donner à l'agent les moyens d'apprendre, partant de zéro, ces comportements. Ceci signifie non seulement apprendre à accomplir une tâche donnée, mais aussi et surtout *déterminer quelles tâches apprendre*.

Ce type d'apprentissage est rendu possible dans le cadre général de l'apprentissage par renforcement grâce à un nouveau mécanisme de sélection d'action (décrit et étudié en détails dans (Buffet *et al.*, 2003)) qui permet à un agent de combiner de manière adaptative des comportements de base en un comportement complexe efficace. Nous montrons que ce même mécanisme peut être utilisé par l'agent pour établir si le nouveau comportement appris, plus complexe, mérite d'être employé comme un *nouveau comportement de base dans le processus de sélection d'action futur*. Ainsi, sans connaissances préalables, notre agent fait croître de manière incrémentale son propre *ensemble* de comportements de base répondant à ses besoins et buts.

1.2. *Cadre de travail*

Afin d'éviter de possibles méprises, clarifions aussi dès à présent les bases de notre architecture d'agent :

– l'agent ne connaît pas de modèle de la dynamique de son environnement et ne l'observe que partiellement. Les buts de l'agent sont représentés par un signal de récompense scalaire dépendant de l'action et de l'état de l'agent ;

– l'agent a un ensemble de comportements de base, chacun répondant à un but/une motivation simple. En outre, chaque comportement de base est de type stimulus-réaction, guidé par la seule observation courante (pas de mémoire à court terme) ;

– dans une situation donnée (perception + choix d'un objectif), l'agent utilise l'information (principalement l'utilité des actions) de ses divers comportements de base pour décider de l'action à effectuer.

1.3. *Exemples*

Voici deux exemples de problèmes auxquels notre travail s'applique. Le premier est un problème applicatif (qui nécessiterait une mise en forme plus complète), et le second servira à valider l'approche proposée.

Assistant personnel intelligent

Un certain nombre de projets actuels visent à rendre notre environnement (maison, ville, entreprise) "intelligent". Un outil privilégié dans ce cadre est l'assistant personnel (PDA), outil informatique portatif que l'on va doter ici de la capacité de communiquer avec d'autres appareils l'environnant.

Au sein d'une maison, l'assistant pourrait être capable d'accomplir des tâches en combinant les capacités de différents appareils (afficher sur la télévision une image stockée dans un appareil photo numérique par exemple). Mais les appareils dispo-

nibles ne sont pas toujours les mêmes d'un instant à l'autre, et l'assistant peut avoir à gérer différentes demandes à la fois.

On se retrouve donc dans une situation avec plusieurs motivations simultanées, avec des objets (appareils) apparaissant/disparaissant de son environnement (selon qu'ils sont utilisés, déplacés...). Cela correspond précisément à la problématique que nous avons définie précédemment.

Le monde des tuiles

Pour illustrer notre approche et tester nos algorithmes, nous emploierons le classique problème du monde des tuiles (voir (Joslin *et al.*, 1993)). Dans celui-ci, tel que représenté sur la figure 3, l'agent doit pousser des tuiles dans des trous tout en évitant d'y tomber lui-même. Intuitivement, il faut pour cela deux comportements de base : pousser des tuiles dans des trous, et éviter des trous.

Avec notre algorithme d'apprentissage, un agent devrait apprendre seul quels comportements de base sont réellement utiles pour résoudre cette tâche complexe.

Dans ce cadre, le monde des tuiles est des plus intéressants car :

– il fait apparaître des comportements aussi bien positifs (essayer de pousser un tuile dans un trou) que négatifs (éviter les trous) ;

– l'un des comportements implique de manipuler plus d'un objet, ce qui évite de se limiter à des interactions trop simples ;

– des objets peuvent avoir plusieurs "rôles" : une tuile peut n'être qu'un obstacle dans certaines situations, ou pourrait être placée dans plusieurs trous disponibles.

Nous verrons tout cela à travers les expérimentations.

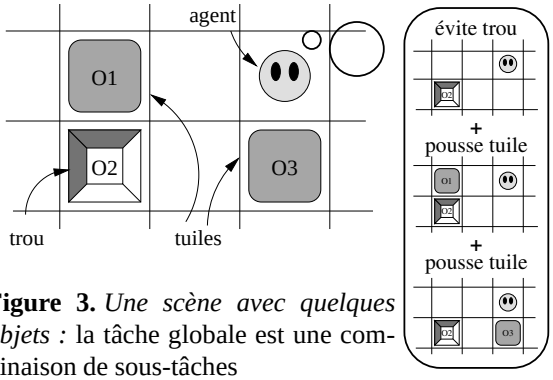


Figure 3. *Une scène avec quelques objets :* la tâche globale est une combinaison de sous-tâches

1.4. *Organisation de l'article*

Pour présenter notre algorithme de définition automatique de comportements de base, nous commençons par définir de manière plus précise la notion de *motivation*

et revenons sur les outils que sont l'apprentissage par renforcement et la sélection d'action (section 2). Ensuite la section 3 introduit la notion de *comportement de base*, avant d'expliquer comment des comportements de base sont combinés en de nouveaux comportements, ces derniers étant des nouveaux comportements de base potentiels. Le processus d'apprentissage d'un ensemble de comportements de base lui-même est présenté dans la section 4, expérimenté et analysé en section 5. Une discussion sur notre algorithme et quelques travaux futurs conclut cet article.

2. Préliminaires

Après avoir introduit quelques notations utiles, cette section présente la notion de motivation, puis revient sur les principaux outils employés dans notre approche : sélection d'action et apprentissage par renforcement.

2.1. *Perception : de l'état à l'observation*

Nous considérons un système constitué d'un agent et de l'environnement dans lequel il évolue, l'environnement étant composé d'objets de types définis et reconnaissables par l'agent. A chaque instant, l'état du système est donné par l'état de l'agent et des objets du système. L'agent ne perçoit que partiellement le système à travers une liste de percepts liés à chaque objet. Ainsi, un agent perçoit la scène courante (la partie du système qui lui est accessible) comme une **observation** qui est un ensemble d'objets **typés** définis par leurs **percepts**.

En notant $\mathfrak{T}$ l'ensemble des types d'objets présents dans le système, nous allons spécifier quelques concepts utilisés par la suite :

– objet perçu obj : défini par son type et la perception courante que l'agent en a : $obj = [t, per]$ où $t \in \mathfrak{T}$. Nous noterons $t(obj)$ et $\mathcal{O}(obj)$ le type et la perception courante de l'objet ;

– observation o : ensemble d'objets (typés) actuellement perçus par l'agent : $o = \{obj_i\}$;

– configuration c : une configuration est un sous-ensemble ordonné d'objets de l'observation. L'ordre est important, car deux objets de même type peuvent avoir des rôles différents ;

– type de configuration C^T : un type de configuration $C^T = \langle t_1, \ldots t_n \rangle$ est une liste de types d'objets (non nécessairement uniques). Cela nous permet de définir un opérateur C^T qui extrait d'une observation donnée o toutes les configurations *compatibles* avec le type de configuration : $\mathcal{C}^T(o) = \{c \subseteq o$ telle qu'il existe une application bijective $m : C^T \rightarrow c\}$;

– observation d'une configuration : liste des perceptions associées avec les objets d'une configuration c : $o(c) = (\mathcal{O}(obj), obj \in c)$.

2.2. *Motivation liée à un comportement*

L'architecture dont nous souhaitons doter notre agent suppose que celui-ci connaisse un certain nombre de comportements dits "de base" répondant chacun à une motivation. En ce sens, il faut en premier lieu définir ce qui constitue une motivation, puisqu'un comportement sera simplement le résultat d'un apprentissage guidé par une motivation.

Un premier aspect est naturellement lié aux signaux de récompense. Pour ne pas être trop restrictifs, une première caractéristique d'une motivation sera un sous-ensemble (non vide) parmi l'ensemble des signaux de récompense élémentaires disponibles. Dans le cas du monde des tuiles, cela revient à choisir un ou deux signaux de récompense parmi les deux disponibles : un pour pousser des tuiles dans des trous (noté $+$) et un pour éviter des trous (noté $-$).

Les comportements que nous pouvons concevoir étant lié à un nombre fixe de percepts *i.e.* d'objets dans notre exemple), il nous faut aussi préciser les types et nombres des objets à prendre en compte (ces types étant instanciés dans un monde donné). Si l'agent ne vise qu'à pousser des tuiles dans des trous, il peut être intéressant pour lui de considérer un trou et deux tuiles simultanément, sans quoi l'une des deux tuiles peut devenir un obstacle non-perçu.

Dans le présent article, une motivation m est ainsi décrite par :

1) $r = \{re_1, \ldots, re_k\}$ un sous-ensemble de signaux de récompense élémentaires et

2) $C^T = \langle t_1, \ldots t_n \rangle$ un type de configuration,

ce qui est représenté sur la figure 4. Cette motivation va guider des approches d'apprentissage par renforcement.

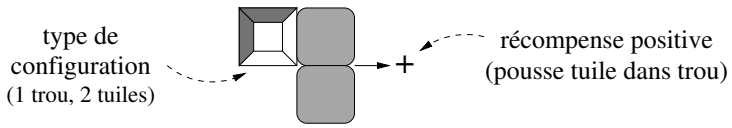


Figure 4. *Une représentation schématique d'une motivation pour 1 trou, 2 tuiles et une récompense positive ($+$) quand une tuile est poussée dans un trou.*
La position relative des objets n'a aucune sorte d'importance. Seuls leur type et leur nombre comptent

2.3. *Sélection d'action*

La sélection d'action pose le problème très général de choisir l'action à accomplir face à différents buts parallèles qui peuvent être hétérogènes et même conflictuels, et répond donc en ce sens à la problématique que nous nous posons. Mais si les

éthologues ont étudié ce problème en profondeur, leurs modèles s'avèrent difficiles à implémenter, comme l'explique Toby Tyrrell (Tyrrell, 1993).

2.3.1. *Problème posé*

Un grand nombre de modèles de sélection d'action mis en œuvre évaluent la qualité d'une action par rapport aux divers buts de haut niveau, se basant sur une somme pondérée de diverses quantités. Le processus obtenu peut être vu comme la consultation de divers experts pour prendre une décision faisant un compromis entre leurs avis (voir figure 5). Dans notre cas, il va falloir identifier dans l'environnement courant les motivations présentes (comme définies dans la section 2.2) pour associer à chacune un comportement de base qui jouera le rôle d'expert.

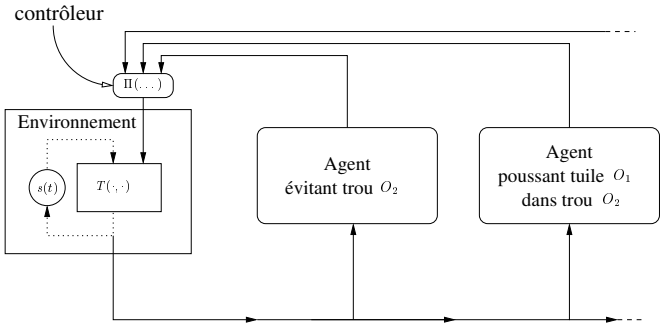


Figure 5. *Une schéma d'architecture de sélection d'action pour le monde des tuiles. Le "contrôleur" doit faire un compromis entre les propositions des différents experts*

Pour éviter la tâche laborieuse de régler manuellement un grand nombre de paramètres associés à ce processus de pesée et de sélection d'actions, diverses approches d'*apprentissage par renforcement* (A/R) ont été étudiées (voir (Lin., 1992, Humphrys, 1996) par exemple). Le présent article se situe dans cette lignée, proposant une approche innovante pour atteindre une plus grande automatisation de la conception de l'agent. Comme nous l'avons déjà dit en introduction, des méthodes existent déjà pour apprendre les comportements de base (l'A/R classique peut suffire) et pour adapter leur combinaison (l'A/R permet aussi d'optimiser les paramètres du contrôleur). Nous proposons en plus de *trouver quels comportements de base sont utiles*.

2.3.2. *Notre cadre de sélection d'action*

Dans sa thèse (Tyrrell, 1993), Toby Tyrrell a fait une analyse des différentes caractéristiques principales des architectures de sélection d'action. Nous en tirons deux choix importants pour notre propre architecture :

– le processus de sélection d'action est de type "flux-libre" : l'action choisie peut être différente des actions recommandées par les comportements de base. L'agent est capable de combiner les comportements en une *nouvelle* décision (ex : une action peut

être classée deuxième par tous les experts, et faire ainsi l'unanimité).
Ceci s'oppose aux processus "winner-takes-all" où *un* expert (donc un comportement) est élu et prend seul la décision ;

– de plus, on ne peut raisonnablement obtenir un contrôleur déterministe, prenant toujours la même décision face à une observation donnée. On risquerait alors trop facilement de rester bloqué dans une situation à laquelle la réponse associée est mauvaise. Il est donc préférable de travailler avec des comportements stochastiques.
Ce problème est similaire à celui des problèmes de décision markoviens *partiellement observables* dans lesquels la propriété de Markov est perdue (voir section 2.4).

2.4. *Notre cadre d'apprentissage par renforcement*

2.4.1. *Problèmes de décision markoviens partiellement observables*

Le problème que nous voulons résoudre dans le cadre de l'apprentissage par renforcement (Kaelbling *et al.,* 1996) peut se formaliser comme un problème de décision markovien partiellement observable (POMDP) (Littman *et al.,* 1995, Cassandra, 1998). Cette généralisation des problèmes de décision markoviens (MDP) (Sutton *et al.,* 1998, Bertsekas *et al.,* 1996, Puterman, 1994) prend explicitement en compte le fait que le système n'est que partiellement perçu par l'agent qui cherche à le contrôler.

Comme schématisé en figure 6, un POMDP est défini par un uplet $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, r \rangle$. Ici, $\mathcal{S}$ est un ensemble fini d'**états** du système, Ω un ensemble fini d'**observations** possibles et $\mathcal{A}$ un ensemble fini d'**actions**. La dynamique du système est décrite par une **fonction de transition** $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ où $\mathcal{T}(s, a, s') = Pr(S_{t+1} = s' | A_t = a, S_t = s)$, et le lien entre état et observation se traduit par la **fonction d'observation** $\mathcal{O}(s, o) = Pr(O_t = o | S_t = s)$. Enfin, r est une application de $\mathcal{S} \times \mathcal{A}$ dans $\mathbb{R}$ qui définit la **récompense** obtenue par le système après chaque transition d'état.

L'agent, qui n'a accès qu'aux observations, doit trouver une **politique** d'action (par exemple sous la forme $\pi : \Omega \rightarrow \Pi(\mathcal{A})$) qui optimise une mesure de performance basée sur la récompense appelée valeur et notée V. Typiquement, la valeur peut être la somme des récompenses sur un horizon fini, la somme pondérée de ces récompenses sur un horizon infini ($\sum_{t=0}^{\infty} \gamma^t r_t$ où $\gamma \in [0, 1)$), ou la récompense moyenne obtenue pendant une transition (*i.e.* pendant un pas de temps).

Dans le cas où l'observation du système est totale (une observation correspond sans équivoque à un état), il a été prouvé qu'il existe au moins une politique optimale déterministe et l'agent peut alors la trouver (Puterman, 1994). Si, comme dans notre cas, l'agent ne connaît pas la dynamique du système, il peut directement apprendre une politique optimale (en utilisant le Q-learning (Watkins, 1989) ou SARSA (Sutton *et al.,* 1998) par exemple) à partir de ses expériences.

Toutefois, dans notre situation, un agent ne perçoit que des observations partielles et, de *son* point de vue, le problème de décision n'est pas markovien. Il peut donc ne

pas y avoir de politique déterministe parmi les solutions optimales. Les algorithmes de programmation dynamique mentionnés précédemment ne sont plus adaptés à ce nouveau cadre, mais peuvent être remplacés par des algorithmes de recherche directe de politiques tels que la descente de gradient en ligne que nous avons utilisée (due à (Baxter *et al.,* 2001a, Baxter *et al.,* 2001b)).

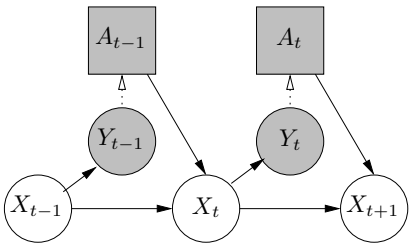


Figure 6. *Modèle graphique d'un POMDP. Les flèches pointillées indiquent que le choix d'une action dépend de l'état courant. Ici, les états sont cachés (représentés avec un fond blanc), contrairement aux observations et actions (fond gris). Les récompenses ne sont pas représentées, afin de garder une figure lisible.*
Note : les décisions prises ne sont ici dépendantes que de l'observation courante

L'apprentissage est d'autant plus difficile dans ces nouvelles conditions, du fait de possibles optima locaux dans lesquels l'agent peut tomber définitivement. Les lecteurs intéressés peuvent se référer à (Littman *et al.,* 1995) et (Singh *et al.,* 1994) sur le sujet de l'utilisation dans un cadre *partiellement* observable d'algorithmes dédiés à un cadre *totalement* observable, et à (Jaakkola *et al.,* 1994) pour une alternative à la descente de gradient.

2.4.2. *La descente de gradient de Baxter et Bartlett*

Sans entrer dans les détails de cette descente de gradient en-ligne, nous décrivons ici succinctement son principe et les conditions dans lesquelles elle a été utilisée.

Comme proposé dans (Baxter *et al.,* 2001a, Baxter *et al.,* 2001b), une politique π dépend d'un ensemble de paramètres Θ. Ceci conduit à une valeur espérée $V(\pi_\Theta) = V(\Theta)$. La descente de gradient permet d'obtenir une politique *localement* optimale en cherchant un Θ^* qui annule le gradient $\nabla V(\Theta)$.

L'ensemble de paramètres que nous choisissons est $\Theta = \{\theta(o,a),\ (o,a) \in \Omega \times \mathcal{A}\}$, où tout $\theta(o,a)$ est un paramètre à valeurs réelles. La politique est définie par les probabilités d'effectuer l'action a quand o est observé par :

$$\pi_\Theta(o,a) = \frac{e^{\theta(o,a)}}{\sum_{b\in\mathcal{A}} e^{\theta(o,b)}}$$

A l'aide de cet algorithme, la donnée d'une motivation (définie en section 2.2) permet d'apprendre la politique qui servira au comportement de base correspondant dans notre mécanisme de sélection d'action (introduit en section 2.3).

3. Combiner des comportements de base

Nous présentons ici brièvement le mécanisme de sélection d'action (apprentissage et combinaison de comportements) utilisé dans cet article. Différentes variantes ont été envisagées, dont une analyse est faite dans (Buffet, 2003) (ainsi qu'une présentation plus complète).

3.1. *Comportements de base*

Pour une motivation donnée, tout algorithme d'A/R adapté aux observations partielles peut être utilisé. Il n'est requis que d'avoir une motivation, c'est à dire :

– de définir une fonction de récompense "guide" (comme la somme de fonctions de récompenses élémentaires sélectionnées), et

– de construire la perception basée sur les observations des objets (objets correspondants au type de configuration).

Le premier résultat obtenu est une politique stochastique (à l'aide de l'algorithme vu en section 2.4.2). Néanmoins, il faudra mémoriser d'autres informations pour faire usage de ce "comportement". De la motivation de départ, on va retenir le type de configuration (qui servira à trouver les objets adéquats dans l'environnement), mais pas la fonction de récompense : on se contentera de la fonction qualité Q (qui aidera à peser l'importance du comportement en fonction de l'espérance de gain).

Un **comportement** b est ainsi défini par un uplet $\langle \mathcal{C}_b^T, P_b, Q_b \rangle$, où :

1) $\mathcal{C}_b^T$ est le type de configuration : l'uplet des types d'objets impliqués dans le comportement ;

2) $P_b(a|o(c))$ est la politique de décision stochastique. A toute observation d'une configuration, elle associe une distribution de probabilité sur les actions ;

3) $Q_b(o(c), a)$ est la Q-table de cette politique, donnant l'espérance de récompense actualisée d'une paire (observation, action).

Un tel comportement répond à une motivation donnée. Mais notre objectif est de trouver un ensemble de comportements qui puissent être combinés ensemble pour prendre une décision. Ces comportements sélectionnés sont nommés **comportements de base**, notés ***bb*** (acronyme de "*basic behavior*") et constituent un ensemble noté $\mathcal{B}$. Avant de voir comment choisir cet ensemble, la prochaine section présente le processus de sélection d'action utilisé.

3.2. *Principe de notre architecture de sélection d'action*

La prise de décision consiste en deux étapes. D'abord, l'agent doit identifier les ensembles d'objets déclenchant des comportements de base. Ensuite, les décisions stochastiques correspondantes sont combinées en une seule faisant un compromis.

La figure 3 illustre ce processus avec seulement deux bb intuitifs :

1) b_p qui considère une tuile et un trou, et la récompense pour pousser une tuile dans un trou, et

2) b_e qui considère un trou, et la récompense pour tomber dans un trou.

On a alors deux raisons pour utiliser b_p : les configurations (uplets d'objets) $cfg_1 = \langle O_1, O_2 \rangle$ et $cfg_2 = \langle O_3, O_2 \rangle$, et une raison pour utiliser b_e : la configuration $cfg_3 = \langle O_2 \rangle$. A chaque paire (bb, cfg) sont associées une distribution de probabilité sur les actions et les Q-valeurs associées, tout cela étant employé pour calculer une distribution de probabilité finale à appliquer, comme montré sur la figure 7.

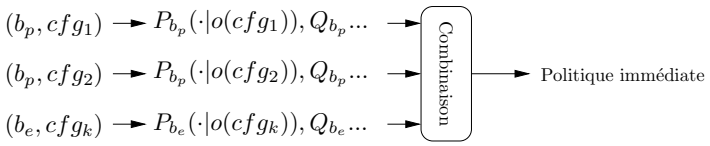


Figure 7. *Comment obtenir une distribution de probabilité sur les actions avec les trois paires (bb, cfg) identifiées*

Les deux sections suivantes décrivent plus en détail les principales opérations que sont la décomposition de la scène et la combinaison de comportement elle-même.

3.2.1. *Décomposition de la scène*

L'objectif de cette décomposition est de trouver quels sont les comportements de base applicables dans une situation donnée.

L'environnement de l'agent est constitué d'objets typés (tels que les trous et tuiles de la figure 3). L'observation de l'agent est un ensemble d'objets perçus dans l'environnement. Chaque comportement de base n'est concerné que par certains types d'objets. Ainsi, ils sont "instanciés" en les associant avec des configurations (cfg), *i.e.* des sous-ensembles d'objets de types appropriés. Cela amène à une décomposition de la scène en un ensemble $\mathcal{BC}(o)$ de paires $(comportement\ de\ base, configuration)$. Revenant à la figure 3, nous avons ici trois telles paires : $(b_e, \langle O_2 \rangle)$, $(b_p, \langle O_1, O_2 \rangle)$ et $(b_p, \langle O_3, O_2 \rangle)$.

Nous avons vu comment chaque scène est décomposée en paires (bb, cfg). On a une politique de décision stochastique ($P_{bb}()$) pour chaque paire. Et, comme chaque

observation est associée avec une observation $o(cfg)$, chaque paire fournit une distribution de probabilité sur les actions $P_{bb}(a|o(cfg))$.

3.2.2. *Combinaison pondérée de comportements de base*

Chaque comportement de base bb de $\mathcal{BC}(o)$ identifié lors de la décomposition de la scène est associé avec une politique d'action $P_{bb}()$ et une qualité $Q_{bb}()$. Avec ces données, l'étape de combinaison consiste à calculer une nouvelle distribution de probabilité sur les actions, déterminant ainsi la politique réelle immédiate de l'agent.

La combinaison peut prendre des formes diverses. Celle que nous avons employée ici dépend d'un ensemble $\Theta = (\theta_1, \cdots, \theta_n)$ de paramètres et est exprimée par la formule suivante (où $\mathcal{C}(b, o)$ est l'ensemble des configurations observées liées à b) :

$$Pr(a|o) = \frac{1}{K}.\frac{1}{k_{(o,a)}} \sum_{b \in \mathcal{B}} \underbrace{e^{\theta_b}}_{\text{à apprendre}} [\underbrace{\sum_{c \in \mathcal{C}(b,o)} |Q_b(o,c,a)|.P_b(o,c,a)}_{\text{déjà connu}}]$$

$$(\text{ avec } k_{(o,a)} = \sum_{(b,c) \in \mathcal{BC}(o)} w_b(o,c,a) = \sum_{(b,c) \in \mathcal{BC}(o)} e^{\theta_b}.|Q_b(o,c,a)| \ )$$

Apprentissage des poids

La politique combinée globale ne dépend que de l'ensemble Θ de paramètres, lesquels sont peu nombreux puisqu'il n'y a qu'**un paramètre pour chaque comportement de base**.

Comme le réglage de ces poids est un problème d'optimisation (maximiser l'espérance du gain moyen), ces paramètres peuvent être appris par renforcement à l'aide d'un recuit simulé. L'algorithme a juste à faire une bonne estimation de l'espérance du gain moyen pour chaque jeu de paramètres.

On notera que ce processus de sélection d'action est "extensible" puisqu'on peut utiliser plusieurs instances d'un même bb sans apprendre de nouveaux paramètres.

3.3. *Problèmes ouverts*

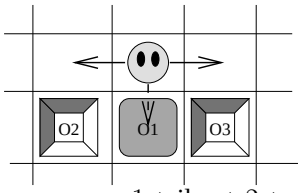
L'algorithme que nous venons de brièvement présenter a été testé sur le monde des tuiles dans (Buffet *et al.*, 2002, Buffet, 2003) en employant deux bb "intuitifs" (`[pousse tuile dans trou]` et `[évite trou]`). Malgré de "bons résultats", l'optimalité n'était pas atteinte et plusieurs limitations étaient citées. Nous insistons sur les suivantes :

– **Optima locaux**. L'apprentissage de paramètres est difficile du fait des nombreux optima locaux vers lesquels le recuit simulé peut converger. De plus, cet apprentissage

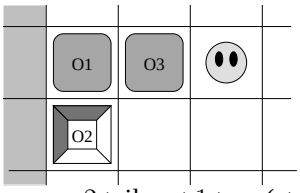
reste long malgré le faible nombre de paramètres à apprendre : il faut prendre le temps de bien évaluer le comportement obtenu avec un jeu de paramètres donné.

– **Méthode de combinaison**. Combiner des avis de divers experts est un problème très ouvert (voir (Genest *et al.*, 1986) par exemple). Les différentes méthodes évaluées dans (Buffet, 2003) reflètent en partie leur diversité et l'attention qu'il faut porter à ce choix. Nous considérons ici qu'une méthode satisfaisante a été choisie.

– **Choix des comportements de base**. L'ensemble des comportements de base disponibles a une grande influence sur les performances de l'algorithme de combinaison. Des bb très généraux (souvent intuitifs) sont requis pour avoir un agent adaptable, mais d'autres, plus spécialisés, permettraient de répondre à des cas difficiles précis (voir les deux blocages de la figure 8). Choisir les comportements de base est précisément le sujet du présent article.



a) Blocage avec 1 tuile et 2 trous : Aucun des comportements de base ne suggère d'aller vers le sud



b) Blocage avec 2 tuiles et 1 trou (et un mur est) : Les deux bb "pousse" suggèrent d'aller à l'ouest, alors que le bon choix serait le nord

Figure 8. *Dans les deux situations a) et b), aucun des comportements de base "intuitifs" employés ne suggère une décision appropriée, d'où un long blocage de l'agent, et donc une grande perte d'efficacité*

4. Quels comportements de base utiliser ?

4.1. *Comment détecter un nouveau comportement de base*

En quelques mots, la construction d'un nouveau comportement de base à partir d'un ensemble existant de bb repose sur trois étapes :

1) apprendre le meilleur comportement combiné pour une motivation donnée,

2) essayer d'améliorer ce comportement et

3) décider s'il sera alors utilisé comme un nouveau comportement de base.

4.1.1. *Apprendre à combiner des comportements de base*

Etant donné un ensemble de comportements de base $\mathcal{B} = \{bb_1, \cdots, bb_n\}$, on sait donc adapter efficacement la formule qui les combine en optimisant les paramètres de réglage θ_b. Le comportement résultant de ce processus d'apprentissage est un *compor-*

tement combiné[1] : $cb = \mathcal{F}(\mathcal{B}, \Theta)$ où $\Theta = (\theta_1, \cdots, \theta_n)$ est un vecteur de paramètres appris de la taille de $\mathcal{B}$.

On constate malheureusement que les choix intuitifs de comportements de base ne permettent pas de répondre à certaines situations de blocage particulières (voir figure 8), lesquelles sont peu nombreuses (faible pourcentage des états possibles), mais font perdre un temps précieux (fort pourcentage du temps de l'agent).

4.1.2. *Amélioration*

Sachant que seules peu de situations posent réellement problème, on en déduit que le *comportement combiné* cb obtenu pour la motivation $m = (r, C^T)$ choisie est très proche d'un comportement très efficace, et n'aurait besoin que de quelques corrections localisées. De là est venue l'idée de partir d'un tel comportement combiné pour apprendre un *nouveau comportement de base* nb, l'apprentissage ayant principalement à corriger les quelques décisions erronées qui font perdre le plus de temps[2].

Pour ce faire, une descente de gradient "par renforcement" en ligne (Baxter *et al.,* 2001b) permet d'améliorer la politique du comportement combiné cb dont nous disposons, même dans une cadre d'A/R non markovien tel que le nôtre. De la nouvelle politique ainsi obtenue, nous pouvons dériver un nouveau comportement (noté nb) avec la motivation et la qualité associées. Ce procédé est résumé dans l'algorithme 1 et illustré par la figure 9.

Algorithme 1 Améliorer un comportement combiné en un nouveau comportement

Entrées: $\mathcal{B}$ un ensemble de comportements de base.
Une motivation $m = (r, C^T)$ (signaux de récompense + type de configuration).

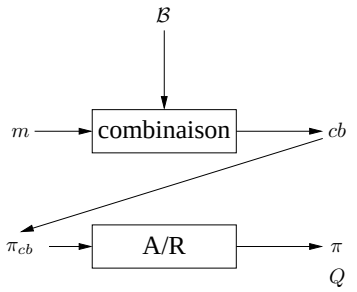
1: Adapter la combinaison des bb courants de $\mathcal{B}$ (pour la motivation m).
Résultat : π_{cb} politique associée à m.
2: Apprendre par renforcement une nouvelle politique pour m, en partant de π_{cb}.
Résultat : π politique améliorée associée à m.
3: En déduire la table de Q-valeurs associée à cette nouvelle politique.
Résultat : Q.

Sorties: Nouveau comportement nb associé à m
(de politique π et de table de Q-valeurs associée Q).

Les expérimentations effectuées (Buffet *et al.,* 2002) montrent qu'il est largement préférable de suivre cet algorithme plutôt que de faire un apprentissage partant de zéro. On peut ainsi dépasser des optima locaux (se contenter d'apprendre à éviter des trous dans notre cas), et l'apprentissage est largement accéléré.

On peut en revanche se demander si l'on n'en apprend pas trop si seules quelques situations sont problématiques. Malheureusement, il n'est pas évident de détecter ces

1. Attention : un comportement combiné n'est pas un comportement comme défini en section 3.1, ce n'est que le résultat d'une combinaison.
2. cb et nb sont des acronymes de l'anglais "combined behavior" et "new behavior".



Etant donné un ensemble de comportements de base $\mathcal{B}$ et une motivation m, on cherche d'abord un "comportement combiné" cb pour n'en garder que la politique π_{cb}. Puis on améliore cette politique par A/R pour obtenir un nouveau comportement de politique π.

Figure 9. *Schéma de principe de l'amélioration d'un comportement combiné*

situations, d'autant que l'on peut avoir des boucles comportementales dont l'agent ne sait sortir. Une solution envisageable serait de comparer π_{cb} avec π pour voir ce qui a été modifié par l'apprentissage par renforcement. Mais un autre aspect à considérer est le calcul des Q-valeurs : peut-on se contenter de retenir les Q-valeurs associées aux décisions nouvelles ? Nous laissons cette question en suspend, nous contentant de ce nouveau comportement complet nb.

4.1.3. *Nouveau comportement de base ? Un critère*

Supposant qu'il n'est pas évident que le nouveau comportement nb soit bien meilleur que le comportement combiné cb, la question qui se pose maintenant est de savoir si l'utilisation de nb comme comportement de base dans le mécanisme de sélection d'action améliorerait le comportement global de l'agent.

Nous proposons ici de baser cette décision sur une comparaison entre la qualité du comportement combiné et celle du nouveau comportement. L'hypothèse suivie est que, si un comportement combiné est nettement amélioré par apprentissage, le nouveau comportement obtenu apporte de nouvelles connaissances utiles.

Si on appelle (V_{cb}) la qualité du comportement combiné et (V_{nb}) la qualité du nouveau comportement, le nouveau comportement est choisi comme nouveau comportement de base (et ajouté à l'ensemble $\mathcal{B}$) si :

$(|V_{cb} - V_{nb}| > \sigma_s) \wedge$
(Permet d'éviter des erreurs dues à des estimations proches.)

$\{ [(V_{cb} < 0) \wedge (V_{nb} < \sigma_- * V_{cb})]$
(Teste s'il y a une amélioration, même faible, d'un résultat.)

$\vee [(V_{cb} \geq 0) \wedge (V_{nb} > \sigma_+ * V_{cb})] \}$
(Teste s'il y a une amélioration majeure d'un résultat positif.)

NOTE. — Le calcul de V est délicat car il consiste en une estimation par méthode de Monte-Carlo (MacKay, 2003, chapitre 29). On laisse l'agent évoluer au sein de son environnement en mesurant au fur et à mesure la récompense moyenne qu'il reçoit. Comme l'agent peut se retrouver bloqué assez longtemps dans certaines situations,

une bonne estimation demande un temps relativement long pour que de tels blocages soient pris en compte à leur juste valeur.

Ce critère dépend de trois paramètres qui ont une grande influence sur l'algorithme général, comme détaillé en section 4.2. Dans nos expérimentations, ils ont été réglés manuellement. Un travail important serait de tester leur validité sur d'autres tâches, de regarder s'ils doivent être beaucoup modifiés d'une tâche à l'autre.

4.2. *Recherche des comportements de base utiles*

On cherche ici à automatiser le choix des comportements de base. Comme expliqué en section 3.3, c'est une question majeure puisque la sélection intuitive de comportements de base "minimalistes" n'est pas suffisante. De plus, cette étape est requise pour avoir un agent réellement autonome, demandant un minimum de travail de conception.

Mais avant de présenter l'algorithme proposé, notons tout d'abord que l'on peut doter l'ensemble des types de configurations d'une relation d'ordre partielle : un type de configuration C_a^T ayant au moins les mêmes types d'objets et en nombre au moins égal qu'un autre type de configuration C_b^T est "plus grand" que ce second type. On peut de même doter d'une relation d'ordre partielle l'ensemble des sous-ensembles de fonctions de récompense élémentaires. Et, de là, on peut déduire une relation d'ordre sur les motivations : $m_a > m_b$ si et seulement si $C_a^T > C_b^T$ et $r_a > r_b$.

4.2.1. *Principe*

Pour rechercher les comportements de base jugés utiles, nous proposons de construire incrémentalement un ensemble de comportements de base. Dans ce but, partant d'un ensemble vide, on y ajoute progressivement des comportements de plus en plus complexes, en utilisant des bb déjà sélectionnés pour en concevoir de nouveaux. Ceci conduit à construire un ensemble de bb de complexité croissante. Or on vient de voir qu'on a une relation d'ordre partielle sur les motivations. Cette relation d'ordre implique que la "complexité" d'un comportement vient de deux sources (liées à la motivation associée à ce comportement) : la taille du type de configuration, et les fonctions de récompense élémentaires choisies.

Pour résumer les principes adoptés, un nouveau type de configuration étant choisi, les comportements correspondant aux possibles combinaisons de récompense sont appris et testés pour déterminer leur intérêt. La figure 10 illustre la progression selon les types de configuration : la complexité croît quand on descend dans l'arbre. Utilisant le processus décrit en section 4.1.3 pour sélectionner un nouveau comportement de base, la section suivante explique l'algorithme de croissance de l'arbre.

4.2.2. *Croissance de l'arbre*

Ne prenant en compte que les types de configurations, la relation d'ordre définie permet de les organiser sous la forme d'un treillis dans lequel les descendants directs

d'un nœud donné ont chacun un type d'objet différent en plus. Mais faire une recherche de comportements de base utiles dans ce treillis pourrait s'avérer très onéreux. On va donc orienter cette recherche en créant progressivement l'arbre[3] dans lequel est menée l'exploration.

On construit donc un arbre d'*exploration* de comportements de complexité croissante selon le nombre d'objets. Chaque nœud de l'arbre est lié à un type de configuration et est en fait un court sous-arbre des combinaisons de récompense possibles. A l'aide d'un parcours en largeur d'abord, on calcule le meilleur comportement pour chaque nœud, sur la base d'une combinaison des bb plus anciens (comme expliqué en section 4.2.1). Chaque nouveau comportement nb est comparé à la combinaison correspondante cb (critère présenté en section 4.1.3) pour déterminer s'il doit être ajouté à l'ensemble des comportements de base, l'arbre étant alors développé à ce nœud par de nouveaux comportements à explorer.

Revenant à l'exemple de la figure 10, l'arbre suit des types de configurations de plus en plus complexes, et à chacun d'eux correspond un sous-arbre des combinaisons de récompenses possibles ("$-$" (respectivement "$+$") pour la récompense négative pour l'évitement de trou (resp. la récompense positive pour pousser une tuile dans un trou), et "$+-$" qui combine les deux).

4.2.3. *Algorithme*

Avant de le tester, voyons une définition formelle de notre méthode à travers l'algorithme 2. Parmi les données requises par cet algorithme, on a :

– p_{max} : Comme une croissance sans fin de l'arbre est possible, ce paramètre limite la profondeur de l'arbre ;

– p : Les fils immédiats d'un nouveau bb ne sont pas les seuls dignes d'intérêt pour des explorations futures. Ainsi, ce second paramètre définit la profondeur de développement d'un nœud (voir la *profondeur de frange* de (McCallum, 1995)) ;

– $Critere(\cdot, \cdot)$: critère d'évaluation des nouveaux comportements (section 4.1.3).

Les expérimentations qui suivent incluent (section 5.2.1) une illustration du comportement de cet algorithme sur le problème du monde des tuiles.

5. Expérimentations

5.1. *Application au monde des tuiles*

5.1.1. *Description du problème*

Le monde des tuiles est un domaine quadrillé dans lequel une case peut contenir un trou, une tuile ou un agent. L'agent (un seul est considéré) doit pousser des tuiles dans des trous aussi souvent que possible, et évite de passer lui-même dans ces trous. Pour

3. On parlera ici abusivement d'arbre même s'il s'agit en fait d'un graphe orienté acyclique.

Algorithme 2 Un arbre croissant de comportements de base

Entrées: $Critere(\cdot,\cdot)$: pour évaluer les nouveaux comportements.
p_{max} : profondeur maximale de l'arbre.
p : profondeur des sous-arbres développés.
1: T arbre vide. $\mathcal{B}$ ensemble vide.
2: Ajouter à T tous les comportements ayant de 0 à p types d'objets.
3: **pour tout** Comportement b encore à visiter (jusqu'à la profondeur $p_{\max}$) **faire**
4: Adapter la combinaison des bb courants de $\mathcal{B}$:
$V_{cb} \leftarrow$ efficacité de ce comportement combiné cb.
5: Apprendre b par une recherche directe de politique (initialisée par cb) :
$V_{nb} \leftarrow$ efficacité de ce nouveau comportement nb.
6: **si** $Critere(V_{cb}, V_{nb}) = vrai$ **alors**
7: Ajouter à T les fils de b ayant jusqu'à p types d'objets en plus.
8: Marquer b comme `[basique]`. L'ajouter à $\mathcal{B}$.
9: **fin si**
10: Marquer b comme `[visité]`.
11: **fin pour**
Sorties: Un ensemble $\mathcal{B}$ de comportements de base retenus.

détailler la simulation, l'agent peut aller librement dans un trou (et en sortir), mais recevra alors une récompense négative. De plus, quand une tuile est poussée dans un trou, tuile et trou disparaissent pour réapparaître autre part sur la grille. Enfin, pour éviter quelques situations bloquantes, trous et tuiles ne peuvent être sur des cases du bord de la grille.

Comme les exemples précédents le montrent (figure 3), une simple décomposition du problème en comportements de base peut être faite intuitivement : 1-`[éviter trou]` $(b_e, \langle trou \rangle)$ et 2- `[pousser tuile dans trou]` $(b_p, \langle tuile,\ trou \rangle)$. Ils seront utilisées comme référence pour notre génération d'un ensemble de bb (noté $\mathcal{B}_{ref}$).

5.1.2. *Perceptions, actions et récompenses de l'agent*

L'agent voit toujours tous les objets de l'environnement. Le principe de localité n'en est pas moins respecté, du fait des perceptions imprécises (agent myope). Pour tout objet O de la scène, la *perception* de O par l'agent donne :

– **direction(**O**)** : donne la direction de l'objet `(N-NE-E-SE-S-SO-O-NO)`, et

– **proche(**O**)** : dit si l'objet est sur le carré de 9 cases centré sur l'agent `(vrai/faux)`.

Les seules *actions* possibles pour un agent sont de se déplacer d'une case vers le nord, le sud, l'est ou l'ouest (il ne peut demander à rester sur une case). Et pour finir, la *récompense* donnée est $+1$ quand une tuile tombe dans un trou, -3 quand l'agent passe dans un trou, et 0 dans tout autre cas.

Revenant à la figure 3, les perceptions des 3 objets sont ici :
O_1 : $proche(O_1) = faux, \quad direction(O_1) = O$
O_2 : $proche(O_2) = faux, \quad direction(O_2) = O$
O_3 : $proche(O_3) = vrai, \quad direction(O_3) = S$

5.1.3. *Méthodologie*

Tout d'abord, précisons que les paramètres apparaissant dans la description de l'algorithme 2 de croissance d'arbre (y compris le critère de la section 4.1.3) ont été réglés manuellement, et définis comme suit :

$$\sigma_s = 500 \qquad \sigma_- = 0,9 \qquad \sigma_+ = 2 \qquad p_{max} = 5 \qquad p = 2$$

Pour analyser le comportement de l'algorithme proposé, les expérimentations conduites prennent deux directions :

1) appliquer l'algorithme de croissance d'arbre pour produire un ensemble de comportements de base $\mathcal{B}_{arbre}$ (dans une grille 6×6) et

2) tester son efficacité dans diverses situations plus complexes (avec plus d'objets, et dans une grille 8×8 pouvant les accueillir).

Les tests conduits dans cette seconde phase consistent à employer la combinaison des bb résultants avec différents nombres de tuiles et de trous (jusqu'à cinq de chaque). La comparaison est alors faite avec les deux bb "intuitifs" de référence (voir section 5.1.1). L'efficacité d'une combinaison est mesurée à travers le gain total reçu pendant $100\,000$ pas de simulation.

5.2. ***Analyse***

5.2.1. *Croissance de l'arbre*

La figure 10 montre l'arbre des comportements évalués qui a été développé par l'algorithme de croissance d'arbre. Les signes mis entre parenthèses indiquent que le comportement lié à ce type de récompense et au type de configuration correspondant a été retenu comme comportement de base.

Profitons d'abord de cette utilisation de l'algorithme sur un exemple pour décrire son fonctionnement (suivre sur la figure 10), sachant que l'arbre n'est que de hauteur 1 (un objet au plus par nœud) au départ.

– L'agent débute sans comportement de base, ce qui résulte en des déplacements aléatoires (mouvement brownien). Placé d'abord dans un environnement sans objets (racine de l'arbre), il n'apprend rien, quels que soient les signaux de récompense considérés : son mouvement brownien fait aussi bien que tout ce qu'il peut apprendre d'autre.

– On place ensuite ce même agent dans un environnement avec un objet. Commençant avec un trou, il va se rendre compte qu'il peut apprendre un comportement utile permettant d'éviter un trou mieux que ne le fait le mouvement brownien (on retient le

comportement et étend l'arbre localement, ajoutant un trou d'un côté, et une tuile de l'autre). Par contre, considérer une seule tuile n'amène rien d'intéressant.

– Les situations à deux objets contiennent toutes deux un trou. Si savoir éviter un trou (le premier *bb* retenu) permet d'en éviter deux, on peut par contre ici apprendre à pousser une tuile dans un trou. Cela donne un second comportement de base (et une nouvelle extension de l'arbre).

– L'agent continue ainsi son apprentissage, ne réussissant à résoudre que l'une des deux situations de blocage à 3 objets...

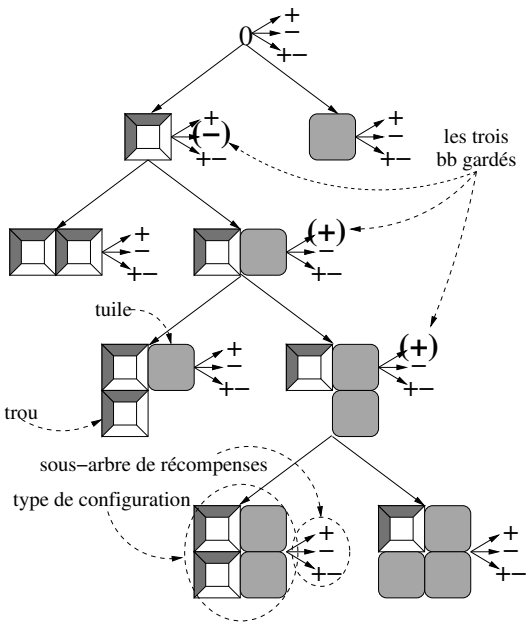


Figure 10. *L'arbre des comportements testés et (entre parenthèses) ceux qui ont été gardés. Les arcs indiquent quels sous-arbres sont développés depuis chaque nœud, sachant que plusieurs nœuds pourraient être à l'origine du développement d'un même sous-arbre*

Les trois *bb* ainsi obtenus (ensemble $\mathcal{B}_{arbre}$) correspondent aux deux *bb* intuitifs (de $\mathcal{B}_{ref}$) avec le comportement [1-trou/2-tuiles avec récompense "$+$"] "spécialisé" dans la résolution d'un blocage (montré figure 8 b)). L'arbre généré semble satisfaisant puisqu'ajouter le comportement "spécialisé" répond à une situation de blocage typique.

Pourtant, le comportement [2-trous/1-tuile, récompense "$+$"] qui résout un autre blocage (figure 8 a)) n'est pas ajouté à l'ensemble des comportements de base. En fait, l'amélioration des performances qu'il aurait amené ne satisfait pas le critère ($V_{nb} \simeq 1,5 * V_{cb} < 2 * V_{cb}$), ce qui explique qu'il soit abandonné. Si l'on utilise un critère "plus souple" ($\sigma_+ = \mathbf{1}, \mathbf{3}$), les deux *bb* spécialisés sont inclus dans $\mathcal{B}$, ainsi que quatre

autres plus complexes. Cela amène d'autres difficultés, le processus d'apprentissage de combinaison étant plus complexe et sujet à tomber dans des optima locaux, même si chacun de ces comportements répond à une situation spécifique intéressante.

5.2.2. *Efficacité de l'arbre*

Pour donner une première idée du problème que pose le monde des tuiles, les figures 11 a) et b) montrent l'efficacité obtenue par un agent se promenant aléatoirement (mouvement brownien) d'une part, et un agent utilisant dans chaque cas un apprentissage par renforcement simple (descente de gradient) d'autre part. Les axes x et y du plan horizontal indiquent le nombre de trous et de tuiles dans les situations évaluées (le nombre d'objets à gérer a contraint à étendre la grille à une taille de 8×8). Sur l'axe z est mesurée l'efficacité dans chaque situation (gain moyen sur 100 000 pas de temps). On constate qu'il s'avère très simple de beaucoup tomber dans des trous sans pousser la moindre tuile dans un trou (agent brownien), et que l'apprentissage par renforcement, à part dans un monde très simple (une tuile et un trou), ne permet que d'éviter les trous.

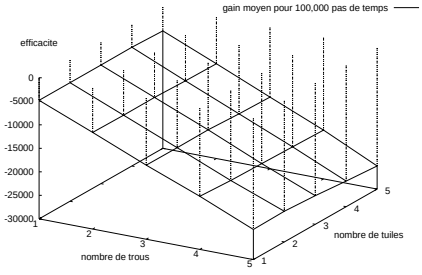
Les figures 11 c) et d) présentent l'efficacité des deux ensembles de comportements de base : $\mathcal{B}_{ref}$ et $\mathcal{B}_{arbre}$, pour pouvoir comparer le second (généré automatiquement par notre algorithme) avec le premier (conçu à la main). Pour certains points, des valeurs numériques sont présentées dans le tableau 1.

A première vue, les deux surfaces sont assez similaires, avec un pic commun dans la situation 1-tuile/1-trou (à laquelle un *bb* "pousser" est dédié). Une amélioration importante ($+30\%$) peut être observée pour 2-tuiles/1-trou, ce qui était attendu puisque $\mathcal{B}_{ref}$ et $\mathcal{B}_{arbre}$ ne diffèrent que par le comportement gérant la motivation "pousser" correspondante. Toutefois, cette amélioration persiste avec un nombre croissant de tuiles à pousser (résultats multipliés par 2) selon l'axe des tuiles (voir figure 12).

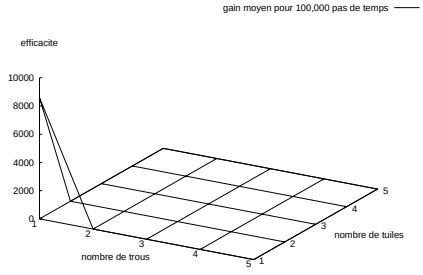
Considérant un nombre croissant de trous, les résultats de la figure 11 d) sont toujours plus proches de la surface de référence 11 c). On a vu qu'un problème lié au critère empêche les améliorations dans ces cas à deux trous et plus. De plus, les variations que l'on peut observer sont partiellement dues à l'estimation de l'efficacité.

5.2.3. *Observation d'un critère plus souple*

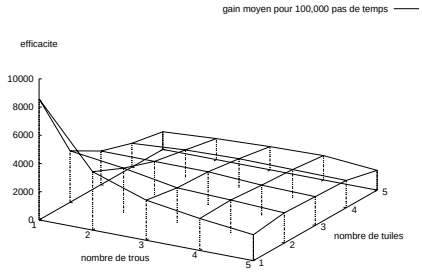
La figure 11 e) montre les résultats obtenus avec le critère plus souple que nous avons déjà mentionné ($V_{nb} > \mathbf{1}, \mathbf{3} * V_{cb}$) et une profondeur maximale de l'arbre $p_{max} = 4$. Ici, le comportement du nœud [2-trous/1-tuile avec récompense "+"] est maintenant retenu dans l'ensemble $\mathcal{B}$. Cela produit sur l'axe des trous un gain similaire à celui observé auparavant sur l'axe des tuiles. Avec ce dernier ensemble $\mathcal{B}'_{arbre}$, l'agent peut même bénéficier des deux améliorations simultanément, comme le montre le tableau 1. Ce même tableau illustre le fait qu'avoir des *bb* inutiles amène une légère perte d'efficacité (voir $\mathcal{B}_{arbre}$ dans les cas 2/1 ou 2/2, ou $\mathcal{B}'_{arbre}$ dans le cas 1/2). Elle s'explique par le fait que ces *bb* inutiles ne sont qu'un bruit ajouté dans la combinaison, perturbant ainsi la prise de décision.



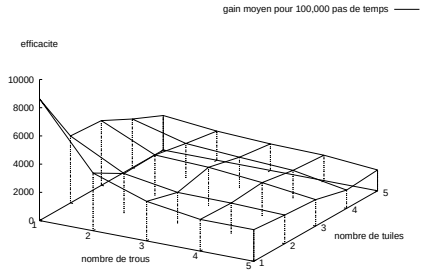
a) déplacement aléatoire (brownien)



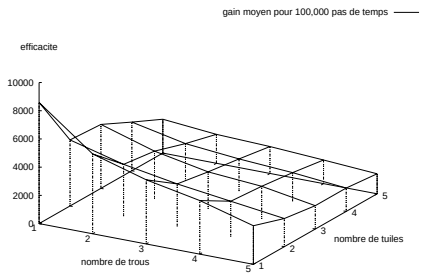
b) apprentissage par renforcement simple



c) comportements de base de $\mathcal{B}_{ref}$



d) comportements de base de $\mathcal{B}_{arbre}$



e) comportements de base de $\mathcal{B}'_{arbre}$

Notes :

– a) $\mathcal{B}_{\emptyset}$= mvt brownien.

– b) Résultat avec un apprentissage par renforcement (descente de gradient) partant de zéro dans chaque cas.

– e) Tests identiques, mais avec $\mathcal{B}'_{arbre}$ = $\mathcal{B}_{arbre}$ + le comportement de base [2-trous/1-tuile avec la récompense "+"].

Figure 11. *Efficacité de la combinaison (avec différents ensembles de comportements de base) ou de l'apprentissage par renforcement dans des situations plus complexes*

$\#_{trous}/\#_{tuiles}$	$\mathcal{B}_{ref}$	$\mathcal{B}_{arbre}$	$\mathcal{B}'_{arbre}$
1 / 2	3648,3	**4747,5**	4680,5
2 / 1	4137,8	4089,6	**5660,8**
2 / 2	3142,3	2807,5	**3673,4**
2 / 3	2369,6	2873,9	**3356,4**
3 / 2	2468,3	2190,3	**3007,7**

Tableau 1. *Comparaison des efficacités des trois ensembles. Avec plus de cinq objets, le gain devient plus difficile à apprécier*

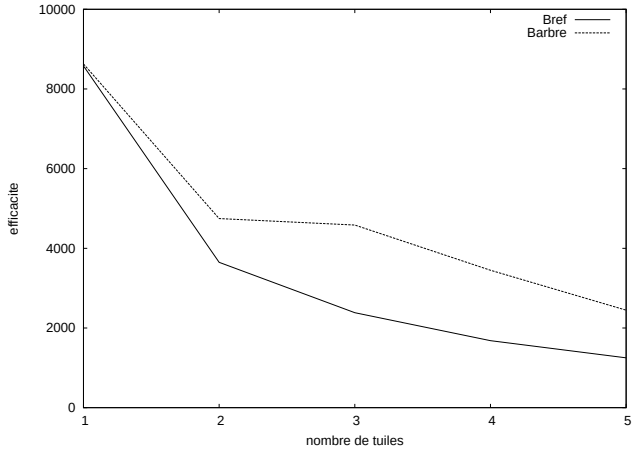
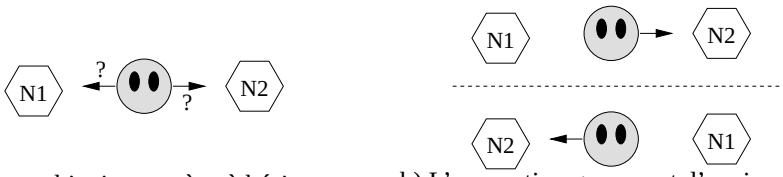


Figure 12. *Comparaison des efficacités de $\mathcal{B}_{ref}$ et $\mathcal{B}_{arbre}$ avec un trou et plusieurs tuiles*

Si la profondeur maximale de l'arbre a été limitée ici à $p_{\max} = 4$, c'est que sans cela l'algorithme trouve de trop nombreux comportements éligibles. En observant les comportements des agents, on se rend compte qu'une combinaison de comportements ne perd pas de temps *que* dans des situations bloquantes telles que celles qu'on a déjà vues, mais aussi dans des situations *indécises*. En effet, si l'agent hésite entre deux objectifs de poids équivalents (deux tuiles possibles à pousser dans le même trou), il risque d'osciller.

Si l'on regarde l'exemple de la figure 13 où un agent peut être attiré par deux sources de nourriture N_1 et N_2, combiner deux instances d'un comportement allant vers une source de nourriture va laisser l'agent indécis, alors qu'apprendre un comportement considérant les *deux* sources de nourriture va lui permettre "d'apprendre à fixer son attention" (toujours sur la source N_2 par exemple).



a) La combinaison amène à hésiter et osciller

b) L'apprentissage permet d'avoir un but persistant

Figure 13. *Apparition d'un phénomène de persistance de l'attention*

6. Discussion, travaux similaires

Si l'on revient aux motivations de notre travail, le but principal de la sélection d'action (être capable de faire face à de multiples tâches) a été atteint. Rappelons que les outils d'apprentissage par renforcement classiques ne s'avèrent pas capables de gérer plus d'une tuile et un trou simultanément (voir figure 11-b) ou les expérimentations dans (Buffet *et al.*, 2002)). Un agent utilisant l'A/R classique ne réussit qu'à éviter de tomber dans les trous, ce qui s'observe par la surface gisant avec un gain nul, mis à part le pic obtenu pour une tuile et un trou.

Avec notre méthode, l'agent reste capable de se comporter correctement avec jusqu'à 5 trous et 5 tuiles. C'est parce qu'elle n'essaye pas de résoudre plusieurs problèmes (répondre à plusieurs motivations) à la fois sans avoir attaqué chaque problème indépendamment dans un premier temps que notre méthode s'en sort. Evidemment, il y a un compromis à trouver entre un nombre réduit de comportements de base (apprentissage plus rapide) et des performances élevées.

Ce qui n'est pas encore automatisé

Le résultat de notre algorithme dépend toujours de paramètres et en particulier d'un critère heuristique. Les paramètres sont liés à l'exploration et à la croissance de l'arbre de comportements. Comme vu en section 5, l'influence du critère peut être décisive. Si l'on conserve ce type de critère, un processus plus intelligent serait d'adapter les paramètres σ pour partir d'un critère souple et le voir se contraindre quand le nombre de comportements de base croît. Ainsi, de nouveaux comportements ne seraient ajoutés que s'ils ont réellement un apport en termes de performances. Bien entendu, des voies d'exploration importantes sont la recherche d'un critère plus pertinent ou celle d'une façon d'automatiser le réglage des paramètres.

Pour toujours considérer des aspects pratiques, notre approche n'appartient pas complètement à l'apprentissage par renforcement *avec* ou *sans* modèle. Si l'algorithme élémentaire d'apprentissage utilisé (la descente de gradient) est dans la classe "sans modèle", l'algorithme de croissance arborescente requiert le passage par un environnement simulé (une forme de modèle), de manière à pouvoir conduire des expérimentations contrôlées. Parmi les apprentissages *avec simulation* (Kearns *et al.*, 2002), on peut citer les travaux de Péret et Garcia (Peret *et al.*, 2004), lesquels ne nécessitent pas de mémoriser un modèle complet (états, transitions...) ou une politique complète, chose qui pourrait s'avérer irréaliste.

Une simulation étant une forme de modèle du système, on se retrouve donc avec le problème d'obtenir un tel modèle dans le cas où l'on travaille sur une application réelle. Il faut alors l'intervention d'un concepteur humain ou l'utilisation d'une méthode d'apprentissage artificiel. On aborde là une autre vaste problématique d'une grande difficulté.

Travaux similaires

Notre approche a été présentée comme une alternative à l'apprentissage par renforcement relationnel dans un cadre partiellement observable. Cette idée pourrait être poussée plus loin en comparant le comportement des deux approches (l'A/R relationnel et la nôtre) sur des problèmes complètement observables. On devrait *a priori* y voir des points communs, mais aussi les limitations de notre cadre plus "souple".

Il serait intéressant de tester et comparer nos travaux sur des projets comparables. Le "house environment" conçu et utilisé dans (Humphrys, 1996) est un exemple qui s'impose, du fait que le travail de Humphrys est aussi l'un des rares à combiner sélection d'action et apprentissage par renforcement. C'est toutefois un problème à la fois :

– plus simple (que notre monde des tuiles) parce qu'il n'y a pas besoin de plusieurs instances du même comportement de base, mais en même temps,

– plus complexe à cause du facteur de branchement élevé qu'il induirait dans l'arbre d'exploration (dû à de nombreux types de récompenses (14) et d'objets (8)).

De premiers résultats montrent que les premiers comportements de base appris sont différents des bb définis manuellement par Humphrys, mais nous n'avons pas les résultats complets pour comparer les performances globales des deux approches.

En élargissant notre champ d'investigation, on peut distinguer deux principales sortes de hiérarchies dans les contrôleurs :

– des hiérarchies "parallèles" comme la nôtre ou toutes celles relevant du domaine de la sélection d'action – plusieurs motivations simultanées guidant l'agent, ce qui implique de faire un compromis – , et

– des hiérarchies "séquentielles" qui ne vont s'attaquer qu'à une tâche élémentaire à la fois, ces tâches (ex : trouver une clef) pouvant être les étapes d'une tâche de plus haut niveau (ouvrir une porte), seule cette dernière amenant une récompense.

Les hiérarchies séquentielles sont plus courantes et mieux maîtrisées dans la mesure où elles n'essayent pas de faire plusieurs choses à la fois. Dans le monde de l'apprentissage par renforcement, on peut citer de nombreux travaux (Mahadevan *et al.*, 1992, Kaelbling, 1993, Lin, 1993, Dayan *et al.*, 1993, Parr, 1998, Hauskretch *et al.*, 1998, Dietterich, 2000). Dans les deux problèmes, on retrouve la même difficulté pour établir la hiérarchie elle-même. Les algorithmes proposés par Digney (Digney, 1998) (nested Q-learning) et Hengst (Hengst, 2002) (HEXQ), par exemple, permettent de construire des hiérarchies séquentielles, ce que nous faisons pour notre part dans le cadre de hiérarchies parallèles.

En fait, dans notre travail, apprendre à la fois des comportements de base et comment les combiner a permis d'accroître l'autonomie de l'agent. Evidemment, la main de l'homme est toujours requise, en particulier pour indiquer les différents types d'objets et de récompenses liés à l'environnement. C'est aussi le cas dans (Digney, 1998), et semble inévitable du point de vue de (Urzelai *et al.*, 1998). Les travaux de ces

derniers visent précisément à trouver le juste équilibre entre conception manuelle et apprentissage artificiel pour concevoir efficacement des agents autonomes. Ils argumentent que l'humain est plus efficace pour définir des comportements de base, même s'ils sont grossiers et raffinés ultérieurement par l'agent. Notre travail, comme ceux de (Digney, 1998) voire de (Nehmzow *et al.*, 1993), semble réduire la tâche du concepteur humain : il n'a à fournir que des types de récompenses et de perceptions. Enfin, de telles approches sont à rapprocher du développement cognitif (Weng, 2002) par leur aspect incrémental (voir aussi chez l'humain (Piaget, 1967)).

7. Conclusion

Notre contribution. Nous avons présenté notre travail sur la conception de comportements complexes pour des agents autonomes. Ce travail se situe dans un cadre de sélection d'action où un comportement complexe est la combinaison de comportements simples. Notre contribution principale concerne la conception automatique d'un ensemble de comportements de base qui serviront à l'algorithme de combinaison. Cette tâche est essentielle à la réalisation de comportements complexes et est aussi une première étape vers un méta-apprentissage : d'une certaine manière, l'agent est capable de sélectionner les capacités dont il a besoin.

L'agent n'a besoin que des signaux de renforcement et des types d'objets présents dans l'environnement pour définir de manière incrémentale des comportements de plus en plus complexes. La valeur ajoutée d'un nouveau comportement est utilisée pour déterminer s'il viendra compléter l'ensemble des comportements de base de l'agent, de manière à servir à l'avenir pour élaborer des comportements plus complexes. Ce processus itératif se termine quand il n'y a plus de comportements à considérer.

Validation. Notre algorithme a été testé sur le problème du monde des tuiles. Il a fourni de bons résultats puisqu'il a été capable de proposer un ensemble de comportements plus complet que l'ensemble intuitif construit à la main. De plus, il faut noter que les problèmes difficiles auxquels il a été confronté (du fait du nombre d'objets présents) ne peuvent être résolus par de l'A/R classique.

Travaux futurs. Un certain nombre d'améliorations et de problèmes ouverts restent en suspend. Il est très important de travailler à un meilleur critère de sélection des comportements "pertinents", le critère actuel étant une heuristique simple. Tester nos algorithmes sur d'autres problèmes est aussi nécessaire pour déterminer plus précisément ses capacités et mieux comprendre l'influence des quelques paramètres de l'algorithme qui sont toujours réglés manuellement. Enfin, comme l'efficacité de l'algorithme de combinaison peut être affecté par la taille de l'ensemble des comportements de base, ce processus de combinaison mériterait d'être encore étudié et amélioré.

La capacité obtenue ici de répondre à plusieurs motivations simultanées pourrait être des plus utiles au sein d'un système multi-agent coopérant (Jennings *et al.*, 1998). En effet, un agent aura la possibilité d'interagir avec l'un ou l'autre de ses congénères, voire avec plusieurs. On retrouve donc clairement ce besoin de pouvoir faire un compromis entre différents objectifs possibles, et d'apprendre des tâches impliquant des nombres variables de perceptions. Malheureusement, l'apprentissage par renforcement dans un cadre multi-agent est en lui-même déjà un problème des plus complexes (Dutech *et al.*, 2001, Shoham *et al.*, 2003). L'aborder avec l'approche présentée dans cet article serait probablement très ambitieux.

Une perspective intéressante serait que l'agent définisse de lui-même des *abstractions d'objets*. Pour l'instant, les types d'objets dans l'environnement sont fournis à l'agent par un concepteur humain, mais nous pensons que la sélection de comportements pourrait servir à catégoriser automatiquement les objets en classes (telles que "obstacles", "but", "à bouger", etc). Ce serait une nouvelle étape vers un vrai méta-apprentissage et une plus grande autonomie des agents.

8. Bibliographie

Baxter J., Bartlett P., « Infinite-Horizon Policy-Gradient Estimation », *Journal of Artificial Intelligence Research*, vol. 15, p. 319-350, 2001a.

Baxter J., Bartlett P., Weaver L., « Experiments with Infinite-Horizon, Policy-Gradient Estimation », *Journal of Artificial Intelligence Research*, vol. 15, p. 351-381, 2001b.

Bertsekas D., Tsitsiklis J., *Neurodynamic Programming*, Athena Scientific, 1996.

Boutilier C., Reiter R., Price B., « Symbolic Dynamic Programming for First-order MDPs », *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, p. 690-697, 2001.

Buffet O., Une double approche modulaire de l'apprentissage par renforcement pour des agents intelligents adaptatifs, PhD thesis, Université Henri Poincaré, Nancy 1, septembre, 2003. Laboratoire Lorrain de recherche en informatique et ses applications (LORIA).

Buffet O., Dutech A., Charpillet F., « Adaptive Combination of Behaviors in an Agent », *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'02)*, 2002.

Buffet O., Dutech A., Charpillet F., « Automatic Generation of an Agent's Basic Behaviors », *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'03)*, 2003.

Cassandra A. R., Exact and Approximate Algorithms for Partially Observable Markov Decision Processes, PhD thesis, Brown University, Department of Computer Science, Providence, RI, 1998.

Dayan P., Hinton G., « Feudal Reinforcement Learning », *Advances in Neural Information Processing Systems 5 (NIPS'93)*, 1993.

Dietterich T., « Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition », *Journal of Artificial Intelligence Research*, vol. 13, p. 227-303, 2000.

Digney B., « Learning Hierarchical Control Structure for Multiple Tasks and Changing Environments », *Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior (SAB'98)*, 1998.

Dutech A., Buffet O., Charpillet F., « Multi-Agent Systems by Incremental Gradient Reinforcement Learning », *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 2001.

Dzeroski S., Raedt L. D., Driessens K., « Relational reinforcement learning », *Machine Learning*, vol. 43, p. 7-52, 2001.

Genest C., Zidek J., « Combining Probability Distributions : A Critique and an Annotated Bibliography », *Statistical Science*, vol. 1, n° 1, p. 114-135, February, 1986.

Gretton C., Thiébaux S., « Exploiting First-Order Regression in Inductive Policy Selection », *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04)*, 2004.

Hauskretch M., Meuleau N., Kaelbling L., Dean T., Boutilier C., « Hierarchical Solution of Markov Decision Processes Using Macro-Actions », *Proceedings of the Fourteenth International Conference on Uncertainty in Artificial Intelligence (UAI'98)*, p. 220-229, 1998.

Hengst B., « Discovering Hierarchy in Reinforcement Learning with HEXQ », *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*, p. 243-250, 2002.

Humphrys M., « Action Selection methods using Reinforcement Learning », *From Animals to Animats 4 : 4th International Conference on Simulation of Adaptive Behavior (SAB-96)*, September, 1996.

Jaakkola T., Jordan M., Singh S., « On the Convergence of Stochastic Iterative Dynamic Programming Algorithms », *Neural Computation*, vol. 6, n° 6, p. 1186-1201, 1994.

Jennings N., Sycara K., Wooldridge M., « A Roadmap of Agent Research and Development », *Autonomous Agents and Multi-Agent Systems*, vol. 1, p. 7-38, 1998.

Joslin D., Nunes A., Pollack M. E., TileWorld Users' Manual, Technical Report n° TR 93-12, August, 1993.

Kaelbling L., « Hierarchical Learning in Stochastic Domains : Preliminary Results », *Proceedings of the Tenth International Conference on Machine Learning (ICML'93)*, 1993.

Kaelbling L., Littman M., Moore A., « Reinforcement Learning : A Survey », *Journal of Artificial Intelligence Research*, vol. 4, p. 237-285, 1996.

Kearns M., Mansour Y., Ng A., « A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes », *Machine Learning*, vol. 49, p. 193-208, 2002.

Lin. L.-J., « Self-improving reactive agent based on reinforcement learning, planning and teaching. », *Machine Learning*, vol. 8, p. 293-321, 1992.

Lin L.-J., « Hierarchical Learning of Robot Skills », *Proceedings of the IEEE International Conference on Neural Networks (ICNN'93)*, 1993.

Littman M., Cassandra A., Kaelbling L., « Learning policies for partially observable environments : scaling up », *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, 1995.

MacKay D., *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

Mahadevan S., Connell J., « Automatic Programming of Behavior-based Robots using Reinforcement Learning », *Artificial Intelligence*, vol. 55, n° 2-3, p. 311-365, June, 1992.

McCallum R. A., Reinforcement Learning with Selective Perception and Hidden State, PhD thesis, University of Rochester, 1995.

Nehmzow U., Smithers T., McGonigle B., « Increasing Behavioural Repertoire in a Mobile Robot », *From Animals to Animats : Proceedings of the Second Conference on the Simulation of Adaptive Behavior (SAB'93)*, 1993.

Parr R. E., Hierarchical Control and Learning for Markov Decision Processes, PhD thesis, 1998.

Peret L., Garcia F., « On-line search for solving Markov Decision Processes via heuristic sampling », *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'2004)*, 2004.

Piaget J., *La Psychologie de l'Intelligence*, Armand Colin, 1967.

Puterman M. L., *Markov Decision Processes–Discrete Stochastic Dynamic Programming*, John Wiley and Sons, Inc., New York, USA, 1994.

Russell S., Norvig P., *Artificial Intelligence : A Modern Approach*, Englewood Cliffs, NJ : prentice Hall, 1995.

Shoham Y., Powers R., Grenager T., Multi-agent reinforcement learning : a critical survey, Technical report, Stanford, 2003.

Singh S., Jaakkola T., Jordan M., « Learning without state estimation in Partially Observable Markovian Decision Processes », *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, 1994.

Sutton R., Barto G., *Reinforcement Learning : an introduction*, Bradford Book, MIT Press, Cambridge, MA, 1998.

Tyrrell T., Computational Mechanisms for Action Selection, PhD thesis, University of Edinburgh, 1993.

Urzelai J., Floreano D., Dorigo M., Colombetti M., « Incremental Robot Shaping », *Connection Science Journal*, 1998.

Watkins C., Learning from delayed rewards, PhD thesis, King's College of Cambridge, UK., 1989.

Weng J., « A Theory of Mentally Developing Robots », *Proceedings of the 2nd International Conference on Development and Learning (ICDL'02)*, june, 2002.

Journal of Artificial Intelligence Research 37 (2010) 329-396 Submitted 08/09; published 3/10

An Investigation into Mathematical Programming for Finite Horizon Decentralized POMDPs

Raghav Aras RAGHAV.ARAS@GMAIL.COM
IMS, Suplec Metz
2 rue Edouard Bélin, Metz Technopole
57070 Metz - France

Alain Dutech ALAIN.DUTECH@LORIA.FR
MAIA - LORIA/INRIA
Campus Scientifique - BP 239
54506 Vandoeuvre les Nancy - France

Abstract

Decentralized planning in uncertain environments is a complex task generally dealt with by using a decision-theoretic approach, mainly through the framework of Decentralized Partially Observable Markov Decision Processes (DEC-POMDPs). Although DEC-POMDPS are a general and powerful modeling tool, solving them is a task with an overwhelming complexity that can be doubly exponential. In this paper, we study an alternate formulation of DEC-POMDPs relying on a *sequence-form* representation of policies. From this formulation, we show how to derive Mixed Integer Linear Programming (MILP) problems that, once solved, give *exact* optimal solutions to the DEC-POMDPs. We show that these MILPs can be derived either by using some combinatorial characteristics of the optimal solutions of the DEC-POMDPs or by using concepts borrowed from game theory. Through an experimental validation on classical test problems from the DEC-POMDP literature, we compare our approach to existing algorithms. Results show that mathematical programming outperforms dynamic programming but is less efficient than forward search, except for some particular problems.
The main contributions of this work are the use of *mathematical programming* for DEC-POMDPs and a better understanding of DEC-POMDPs and of their solutions. Besides, we argue that our alternate representation of DEC-POMDPs could be helpful for designing novel algorithms looking for approximate solutions to DEC-POMDPs.

1. Introduction

The framework of Decentralized Partially Observable Markov Decision Processes (DEC-POMDPs) can be used to model the problem of designing a system made of autonomous agents that need to coordinate in order to achieve a joint goal. Solving DEC-POMDPs is an untractable task as they belong to the class of NEXP-complete problems (see Section 1.1). In this paper, DEC-POMDPs are reformulated into *sequence-form DEC-POMDPs* so as to derive Mixed Integer Linear Programs that can be solved using very efficient solvers in order to design *exact* optimal solutions to finite-horizon DEC-POMDPs. Our main motivation is to investigate the benefits and limits of this novel approach and to get a better understanding of DEC-POMDPs (see Section 1.2). On a practical level, we provide new algorithms and heuristics for solving DEC-POMDPs and evaluate them on classical problems (see Section 1.3).

1.1 Context

One of the main goals of Artificial Intelligence is to build artificial agents that exhibit intelligent behavior. An agent is an entity situated in an environment which it can perceive through sensors and act upon using actuators. The concept of planning, *i.e.*, to select a sequence of actions in order to reach a goal, has been central to the field of Artificial Intelligence for years. While the notion of "intelligent behavior" is difficult to assess and to measure, we prefer to refer to the concept of "rational behavior" as formulated by Russell and Norvig (1995). As a consequence, the work presented here uses a decision-theoretic approach in order to build agents that take optimal actions in an uncertain and partially unknown environment.

We are more particularly interested in cooperative multi-agent systems where multiple independent agents with limited perception of their environment must interact and coordinate in order to achieve a joint task. No central process with a full knowledge of the state of the system is there to control the agents. On the contrary, each agent is an autonomous entity that must execute its actions by itself. This setting is both a blessing, as each agent should ideally deal with a small part of the problem, and a curse, as coordination and cooperation are harder to develop and to enforce.

The decision-theoretic approach to rational behavior relies mostly on the framework of Markov Decision Processes (MDP) (Puterman, 1994). A system is seen as a sequence of discrete states with stochastic dynamics, some particular states giving a positive or negative reward. The process is divided into discrete decision periods; the number of such periods is called the *horizon* of the MDP. At each of these periods, an action is chosen which will influence the transition of the process to its next state. By using the right actions to influence the transition probabilities between states, the objective of the controller of the system is to maximize its long term return, which is often an additive function of the reward earned for the given horizon. If the controller knows the dynamics of the system, which is made of a transition function and of a reward function, algorithms derived from the field of Dynamic Programming (see Bellman, 1957) allow the controller to compute an optimal deterministic *policy*, *i.e.*, a decision function which associates an "optimal" action to every state so that the expected long term return is optimal. This process is called *planning* in the MDP community.

In fact, using the MDP framework, it is quite straightforward to model a problem with one agent which has a full and complete knowledge of the state of the system. But agents, and especially in a multi-agent setting, are generally not able to determine the complete and exact state of the system because of noisy, faulty or limited sensors or because of the nature of the problem itself. As a consequence, different states of the system are observed as similar by the agent which is a problem when different optimal actions should be taken in these states; one speaks then of *perceptual aliasing*. An extension of MDPs called Partially Observable Markov Decisions Processes (POMDPs) deals explicitly with this phenomenon and allows a single agent to compute plans in such a setting provided it knows the conditional probabilities of observations given the state of the environment (Cassandra, Kaelbling, & Littman, 1994).

As pointed out by Boutilier (1996), multi-agent problems could be solved as MDPs if considered from a centralized point of view for planning *and* control. Here, although

planning is a centralized process, we are interested in decentralized settings where every agent executes its own policy. Even if the agents could instantly communicate their observation, we consider problems where the joint observation resulting from such communications would still not be enough to identify the state of the system. The framework of Decentralized Partially Observable Markov Decision Processes (DEC-POMDP) proposed by Bernstein, Givan, Immerman, and Zilberstein (2002) takes into account decentralization of control and partial observability. In a DEC-POMDP, we are looking for optimal *joint policies* which are composed of one policy for each agent, these individual policies being computed in a centralized way but then independently executed by the agents.

The main limitation of DEC-POMDPs is that they are provably untractable as they belong to the class of NEXP-complete problems (Bernstein et al., 2002). Concretely, this complexity result implies that, in the worst case, finding an optimal joint policy of a finite horizon DEC-POMDP requires time that is *exponential* in the horizon if one always make good choices. Because of this complexity, there are very few algorithms for finding exact optimal solutions for DEC-POMDPs (they all have a doubly exponential complexity) and only a few more that look for approximate solutions. As discussed and detailed in the work of Oliehoek, Spaan, and Vlassis (2008), these algorithms follow either a dynamic programming approach or a forward search approach by adapting concepts and algorithms that were designed for POMDPs.

Yet, the concept of decentralized planning has been the focus of quite a large body of previous work in other fields of research. For example, the *Team Decision Problem* (Radner, 1959), later formulated as a Markov system in the field of control theory by Anderson and Moore (1980), led to the *Markov Team Decision Problem* (Pynadath & Tambe, 2002). In the field of mathematics, the abundant literature on Game Theory brings a new way for looking at multi-agent planning. In particular, a DEC-POMDP with finite horizon can be thought as a *game in extensive form with imperfect information and identical interests* (Osborne & Rubinstein, 1994).

Taking inspiration from the field of game theory and mathematical programming to design exact algorithms for solving DEC-POMDPs is precisely the subject of our contribution to the field of decentralized multi-agent planning.

1.2 Motivations

The main objective of our work is to investigate the use of mathematical programming, more especially mixed-integer linear programs (MILP) (Diwekar, 2008), for solving DEC-POMDPs. Our motivation relies on the fact that the field of linear programming is quite mature and of great interest to the industry. As a consequence, there exist many efficient solvers for mixed-integer linear programs and we want to see how these efficient solvers perform in the framework of DEC-POMDPs.

Therefore, we have to reformulate a DEC-POMDP to solve it as a mixed-integer linear program. As shown in this article, two paths lead to such mathematical programs, one grounded on the work from Koller, Megiddo, and von Stengel (1994), Koller and Megiddo (1996) and von Stengel (2002), and another one grounded on combinatorial considerations. Both methods rely on a special reformulation of DEC-POMDPs in what we have called

sequence-form DEC-POMDPs where a policy is defined by the histories (*i.e.*, sequences of observations and actions) it can generate when applied to the DEC-POMDP.

The basic idea of our work is to select, among all the histories of the DEC-POMDP, the histories that will be part of the optimal policy. To that end, an optimal solution to the MILP presented in this article will assign a positive weight to each history of the DEC-POMDP and every history with a non-negative weight will be part of the optimal policy to the DEC-POMDP. As the number of possible histories is exponential in the horizon of the problem, the complexity of a naive search for the optimal set of histories is doubly exponential. Therefore, our idea appears untractable and useless.

Nevertheless, we will show that combining the efficiency of MILP solvers with some quite simple heuristics leads to exact algorithms that compare quite well to some existing exact algorithms. In fact, sequence-form DEC-POMDPs only need a memory space exponential in the size of the problem. Even if solving MILPs can also be exponential in the size of the MILP and thus leads to doubly exponential complexity for sequence-form based algorithms, we argue that sequence-form MILPs compare quite well to dynamic programming thanks to optimized industrial MILP solvers like "Cplex".

Still, our investigations and experiments with Mathematical Programming for DEC-POMDPs do not solely aim at finding exact solutions to DEC-POMDPs. Our main motivation is to have a better understanding of DEC-POMDPs and of the limits and benefits of the mathematical programming approach. We hope that this knowledge will help deciding to what extent mathematical programming and sequence-form DEC-POMDPs can be used to design novel algorithms that look for *approximate* solutions to DEC-POMDPs.

1.3 Contributions

In this paper we develop new algorithms in order to find exact optimal joint policies for DEC-POMDPs. Our main inspiration comes from the work of Koller, von Stegel and Megiddo that shows how to solve games in extensive form with imperfect information and identical interests, that is how to find a *Nash equilibrium* for this kind of game (Koller et al., 1994; Koller & Megiddo, 1996; von Stengel, 2002). Their algorithms caused a breakthrough as the memory space requirement of their approach is linear in the size of the game whereas more canonical algorithms required space that is exponential in the size of the game. This breakthrough is mostly due to the use of a new formulation of a policy in what they call a *sequence-form*.

Our main contribution, as detailed in Section 3.3, is then to adapt the **sequence-form** introduced by Koller, von Stegel and Megiddo to the framework of DEC-POMDPs (Koller et al., 1994; Koller & Megiddo, 1996; von Stengel, 2002). As a result, it is possible to formulate the resolution of a DEC-POMDP as a special kind of mathematical program that can still be solved quite efficiently: a mixed linear program where some variables are required to be either 0 or 1. The adaptation and the resulting mixed-integer linear program is not straightforward. In fact, Koller, von Stegel and Megiddo could only find *one* Nash equilibrium in a 2-agent game. What is needed for DEC-POMDPs is to find the set of policies, called a joint policy, that corresponds to the Nash equilibrium with the highest value, finding "only" one Nash equilibrium – already a complex task – is not enough. Besides, whereas Koller, von Stegel and Megiddo algorithms could only be applied

to 2-agent games, we extend the approach so as to solve DEC-POMDPs with an *arbitrary number of agents*, which constitutes an important contribution.

In order to formulate DEC-POMDPs as MILPs, we analyze in detail the structure of an optimal joint policy for a DEC-POMDP. A joint policy in sequence-form is expressed as a set of individual policies that are themselves described as a set of possible trajectories for each of the agents of the DEC-POMDP. Combinatorial considerations on these individual histories, as well as constraints that ensure these histories do define a valid joint policy are at the heart of the formulation of a DEC-POMDP as a mixed linear program, as developped in Sections 4 and 5. Thus, another contribution of our work is a better understanding of the properties of optimal solutions to DEC-POMDPs, a knowledge that might lead to the formulation of new approximate algorithms for DEC-POMDPs.

Another important contribution of this work is that we introduce **heuristics** for boosting the performance of the mathematical programs we propose (see Section 6). These heuristics take advantage of the succinctness of the DEC-POMDP model and of the knowledge acquired regarding the structure of optimal policies. Consequently, we are able to reduce the size of the mathematical programs (resulting also in reducing the time taken to solve them). These heuristics constitute an important pre-processing step in solving the programs. We present two types of heuristics: the elimination of *extraneous* histories which reduces the size of the mixed integer linear programs and the introduction of *cuts* in the mixed integer linear programs which reduces the time taken to solve a program.

On a more practical level, this article presents three different **mixed integer linear programs**, two are more directly derived from the work of Koller, von Stegel and Megiddo (see Table 4 and 5) and a third one is based solely on combinatorial considerations on the individual policies and histories (see Table 3). The theoretical validity of these formulations is backed by several theorems. We also conducted experimental evaluations of our algorithms and of our heuristics on several classical DEC-POMDP problems. We were thus able to confirm that our algorithms are quite comparable to dynamic programming exact algorithms but outperformed by forward search algorithms like GMAA* (Oliehoek et al., 2008). On some problems, though, MILPs are indeed faster by one order of magnitude or two than GMAA*.

1.4 Overview of this Article

The remainder of this article is organized as follows. Section 2 introduces the formalism of DEC-POMDP and some background on the classical algorithms, usually based on dynamic programing. Then we expose our reformulation of the DEC-POMDP in sequence-form in Section 3 where we also define various notions needed by the sequence-form. In Section 4, we show how to use combinatorial properties of the sequence-form policies to derive a first mixed integer linear program (MILP, in Table 3) for solving DEC-POMDP. By using game theoretic concepts like Nash equilibrium, we take inspiration from previous work on games in extensive form to design two other MILPs for solving DEC-POMDP (Tables 4, 5). These MILPs are smaller in size and their detailed derivation is presented in Section 5. Our contributed heuristics to speed up the practical resolutions of the various MILPs make up the core of Section 6. Section 7 presents experimental validations of our MILP-based algorithms on classical benchmarks of the DEC-POMDP literature as well as on randomly

built problems. Finally, Section 8 analyzes and discusses our work and we conclude this paper with Section 9.

2. Dec-POMDP

This section gives a formal definition of Decentralized Partially Observed Markov Decision Processes as introduced by Bernstein et al. (2002). As described, a solution of a DEC-POMDP is a policy defined on the space of information sets that has an optimal value. This sections ends with a quick overview of the classical methods that have been developed to solve DEC-POMDPs.

2.1 Formal Definition

A **DEC-POMDP** is defined as a tuple $\mathcal{D} = \langle\ I,\ S,\ \{A_i\},\ \mathbb{P},\ \{O_i\},\ \mathbb{G},\ R,\ T,\ \alpha\ \rangle$ where:

- $I = \{1, 2, \cdots, n\}$ is a set of *agents*.
- S is a finite set of *states*. The set of probability distributions over S shall be denoted by $\Delta(S)$. Members of $\Delta(S)$ shall be called *belief states*.
- For each agent $i \in I$, A_i is a set of *actions*. $A = \times_{i \in I} A_i$ denotes the set of joint actions.
- $\mathbb{P} : S \times A \times S \to [0,\ 1]$ is a *state transition function*. For each $s, s' \in S$ and for each $a \in A$, $\mathbb{P}(s, a, s')$ is the probability that the state of the problem in a period t is s' if, in period $t-1$, its state was s and the agents performed the joint action a. Thus, for any time period $t \geq 2$, for each pair of states $s, s' \in S$ and for each joint action $a \in A$, there holds:

  $$\mathbb{P}(s, a, s') \quad = \quad \Pr(s^t = s' | s^{t-1} = s, a^t = a).$$

  Thus, $(S,\ A,\ \mathbb{P})$ defines a discrete-state, discrete-time controlled Markov process.
- For each agent $i \in I$, O_i is a set of *observations*. $O = \times_{i \in I} O_i$ denotes the set of joint observations.
- $\mathbb{G} : A \times S \times O \to [0,\ 1]$ is a *joint observation function*. For each $a \in A$, for each $o \in O$ and for each $s \in S$, $\mathbb{G}(a, s, o)$ is the probability that the agents receive the joint observation o (that is, each agent i receives the observation o_i) if the state of the problem in that period is s and if in the previous period the agents took the joint action a. Thus, for any time period $t \geq 2$, for each joint action $a \in A$, for each state $s \in S$ and for each joint observation $o \in O$, there holds:

  $$\mathbb{G}(a, s, o) \quad = \quad \Pr(o^t = o | s^t = s, a^{t-1} = a).$$

- $R : S \times A \to \mathbb{R}$ is a *reward function*. For each $s \in S$ and for each $a \in A$, $R(s, a) \in \mathbb{R}$ is the reward obtained by the agents if they take the joint action a when the state of the process is s.

- T is the *horizon* of the problem. The agents are allowed T joint-actions before the process halts.
- $\alpha \in \Delta(S)$ is the *initial state* of the DEC-POMDP. For each $s \in S$, $\alpha(s)$ denotes the probability that the state of the problem in the first period is s.

As said, S, A and $\mathbb{P}$ define a *controlled Markov Process* where the next state depends only on the previous state and on the joint action chosen by the agents. But the agents do not have access to the state of the process and can only rely on observations, generally partial and noisy, of this state, as specified by the observation function $\mathbb{G}$. From time to time, agents receive a non-zero reward according to the reward function R.

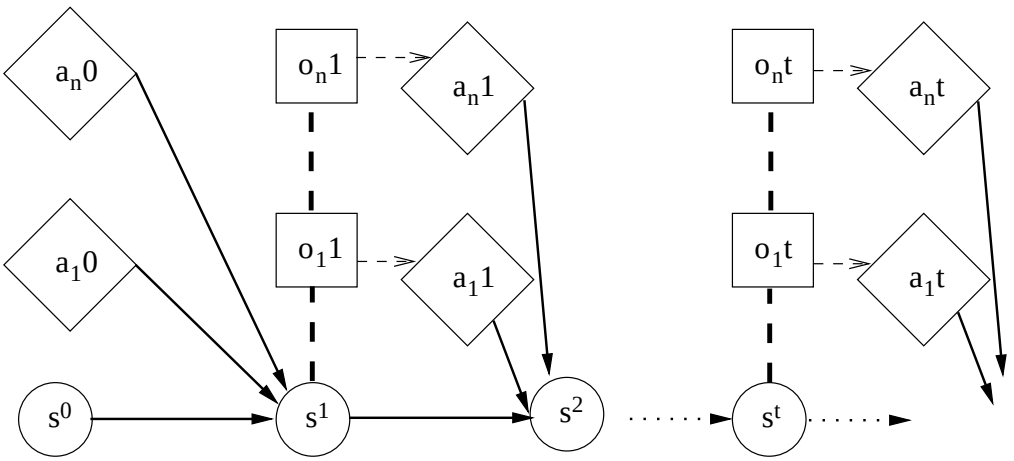


Figure 1: **DEC-POMDP**. At every period t of the process, the environment is in state s^t, every agent i receives observations o_i^t and decides of its action a_i^t. The joint action $\langle a_1^t, a_2^t, \cdots, a_n^t \rangle$ alters the state of the process.

More specifically, as illustrated in Figure 1, the control of a DEC-POMDP by the n agents unfolds over discrete time periods, $t = 1, 2, \cdots, T$ as follows. In each period t, the process is in a state denoted by s^t from S. In the first period $t = 1$, the state s^1 is chosen according to α and the agents take actions a_i^1. In each period $t > 1$ afterward, each agent $i \in I$ takes an action denoted by a_i^t from A_i according to the agent's *policy*. When the agents take the joint action $a^t = \langle a_1^t, a_2^t, \cdots, a_n^t \rangle$, the following events occur:

1. The agents all obtain the same reward $R(s^t, a^t)$.
2. The state s^{t+1} is determined according to the function $\mathbb{P}$ with arguments s^t and a^t.
3. Each agent $i \in I$ receives an observation o_i^{t+1} from O_i. The joint observation $o^{t+1} = \langle o_1^{t+1}, o_2^{t+1}, \cdots, o_n^{t+1} \rangle$ is determined by the function $\mathbb{G}$ with arguments s^{t+1} and a^t.
4. The period changes from t to $t + 1$.

In this paper, the DEC-POMDP we are interested in have the following properties:

- the horizon T is finite and known by the agents;
- agents cannot infer the exact state of the system from their joint observations (this is the more general setting of DEC-POMDPs);
- agents do not observe actions and observations of the other agents. They are only aware of their own observations and reward;
- agents have a perfect memory of their past; they can base their choice of action on the sequence of past actions and observations. We speak of *perfect recall* setting;
- transition and observation functions are stationary, meaning that they do not depend on the period t.

Solving a DEC-POMDP means finding the agents' *policies* (*i.e., their decision functions*) to optimize a given criterion based on the rewards received. The criterion we will work with is called the **cumulative reward** and defined by:

$$\mathbb{E}\left[\sum_{t=1}^{T} R(s^t, \langle a_1^t, a_2^t, \ldots, a_n^t\rangle)\right] \tag{1}$$

where $\mathbb{E}$ is the mathematical expectation.

2.2 Example of DEC-POMDP

The problem known as the "Decentralized Tiger Problem" (hereby denoted MA-Tiger), introduced by Nair, Tambe, Yokoo, Pynadath, and Marsella (2003), has been widely used to test DEC-POMDPs algorithms. It is a variation of a problem previously introduced for POMDPs (*i.e.*, DEC-POMDPs with one agent) by Kaelbling, Littman, and Cassandra (1998).

In this problem, we are given two agents confronted with two closed doors. Behind one door is a tiger, behind the other an escape route. The agents do not know which door leads to what. Each agent, independently of the other, can open one of the two doors or listen carefully in order to detect the tiger. If either of them opens the wrong door, the lives of both will be imperiled. If they both open the escape door, they will be free. The agents have a limited time in which to decide which door to open. They can use this time to gather information about the precise location of the tiger by listening carefully to detect the location of the tiger. This problem can be formalized as a DEC-POMDP with:

- two states as the tiger is either behind the left door (s_l) or the right door (s_r);
- two agents, that must decide and act;
- three actions for each agent: open the left door (a_l), open the right door (a_r) and listen (a_o);
- two observations, as the only thing the agent can observe is that they hear the tiger on the left (o_l) or on the right (o_r).

The initial state is chosen according to a uniform distribution over S. As long as the door remains closed, the state does not change but, when one door is opened, the state is reset to either s_l or s_r with equal probability. The observations are noisy, reflecting the difficulty of detecting the tiger. For example, when the tiger is on the left, the action a_o produces an observation o_l only 85% of the time. So if both agents perform a_o, the joint observation (o_l,o_l) occurs with a probability of $0.85 \times 0.85 = 0.72$. The reward function encourages the agents to coordinate their actions as, for example, the reward when both open the escape door (+20) is bigger than when one listens while the other opens the good door (+9). The full state transition function, joint observation function and reward function are described in the work of Nair et al. (2003).

2.3 Information Sets and Histories

An **information set** φ of agent i is a sequence $(a^1.o^2.a^2.o^3 \cdots .o^t)$ of even length in which the elements in odd positions are actions of the agent (members of A_i) and those in even positions are observations of the agent (members of O_i). An information set of length 0 shall be called the **null information set**, denoted by $\varnothing$. An information set of length $T-1$ shall be called a **terminal information set**. The set of information sets of lengths less than or equal to $T-1$ shall be denoted by Φ_i.

We define a **history** of agent $i \in I$ to be a sequence $(a^1.o^2.\ a^2.\ o^3 \cdots .o^t.a^t)$ of odd length in which the elements in odd positions are actions of the agent (members of A_i) and those in even positions are observations of the agent (members of O_i). We define the **length** of a history to be the number of actions in the history (t in our example). A history of length T shall be called a **terminal history**. Histories of lengths less than T shall be called **non-terminal** histories. The history of null length shall be denoted $\varnothing$. The information set associated to an history h, denoted $\varphi(h)$, is the information set composed by removing from h its last action. If h is a history and o an observation, then $h.o$ is an information set.

We shall denote by $\mathcal{H}_i^t$ the set of all possible histories of length t of agent i. Thus, $\mathcal{H}_i^1$ is just the set of actions A_i. We shall denote by $\mathcal{H}_i$ the set of histories of agent i of lengths less than or equal to T. The size n_i of $\mathcal{H}_i$ is thus:

$$n_i = |\mathcal{H}_i| \quad = \sum_{t=1}^{T} |A_i|^t |O_i|^{t-1} \quad = |A_i| \frac{(|A_i||O_i|)^T - 1}{|A_i||O_i| - 1}. \tag{2}$$

The set $\mathcal{H}_i^T$ of terminal histories of agent i shall be denoted by $\mathcal{E}_i$. The set $\mathcal{H}_i \backslash \mathcal{H}_i^T$ of non-terminal histories of agent i shall be denoted by $\mathcal{N}_i$.

A tuple $\langle h_1, h_2, \ldots, h_n \rangle$ made of one history for each agent is called a **joint history**. The tuple obtained by removing the history h_i from the joint history h is noted h_{-i} and called an i-**reduced joint history**.

Example Coming back to the MA-Tiger example, a set of valid histories could be: $\varnothing$, (a_o), $(a_o.o_l.a_o)$, $(a_o.o_r.a_o)$, $(a_o.o_l.a_o.o_l.a_o)$, $(a_o.o_l.a_o.o_r.a_r)$, $(a_o.o_r.a_o.o_l.a_o)$ and $(a_o.o_r.a_o.o_r.a_r)$. Incidently, this set of histories corresponds to the support of the policy (*i.e.*, the histories generated by using this policy) of the Figure 2, as explained in the next section.

2.4 Policies

At each period of time, a policy must tell an agent what action to choose. This choice can be based on whatever past and present knowledge the agent has about the process at time t. One possibility is to define an **individual policy** π_i of agent i as a mapping from information sets to actions. More formally:

$$\pi_i \quad : \quad \Phi_i \longrightarrow \Delta(A_i) \tag{3}$$

Among the set Π of policies, three families are usually distinguished:

- *Pure policies.* A pure or deterministic policy maps a given information set to *one* unique action. The set of pure policies for the agent i is denoted $\hat{\Pi}$. Pure policies could also be defined using trajectories of past observations only since actions, which are chosen deterministically, can be reconstructed from the observations.
- *Mixed policies.* A mixed policy is a probability distribution over the set of pure policies. Thus, an agent using a mixed policy will control the DEC-POMDP by using a pure policy randomly chosen from a set of pure policies.
- *Stochastic policies.* A stochastic policy is the more general formulation as it associates a probability distribution over actions to each history.

If we come back to the MA-Tiger problem (Section 2.2), Figure 2 gives a possible policy for a horizon 2. As shown, a policy is classically represented by an action-observation tree. In that kind of tree, each branch is labelled by an observation. For a given sequence of past observations, one starts from the root node and follows the branches down to an action node. This node contains the action to be executed by the agent when it has seen this sequence of observations.

Observation sequence	$\varnothing$	o_l	o_r	$o_l.o_l$	$o_l.o_r$	$o_r.o_l$	$o_r.o_r$
Chosen action	a_o	a_o	a_o	a_l	a_o	a_o	a_r

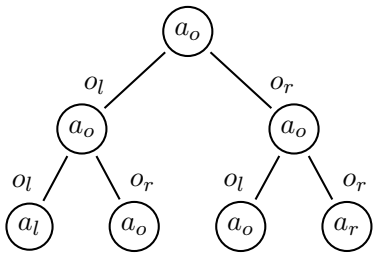


Figure 2: **Pure policy for MA-Tiger**. A pure policy maps sequences of observations to actions. This can be represented by an action-observation tree.

A **joint policy** $\pi = \langle \pi_1, \pi_2, \cdots, \pi_n \rangle$ is an n-tuple where each π_i is a policy for agent i. Each of the individual policies must have the same horizon. For an agent i, we also define the notion of an **i-reduced joint policy** $\pi_{-i} = \langle \pi_1, \cdots, \pi_{i-1}, \pi_{i+1}, \cdots, \pi_n \rangle$ composed of the policies of all the other agents. We thus have that $\pi = \langle \pi_i, \pi_{-i} \rangle$.

2.5 Value Function

When executed by the agents, every T-horizon joint policy generates a probability distribution over the possible sequences of reward from which one can compute the **value** of the policy according to Equation 1. Thus the value of the joint policy π is formally defined as:

$$V(\alpha, \pi) = \mathbb{E}\left[\sum_{t=1}^{T} R(s^t, a^t)|\pi, \alpha\right] \tag{4}$$

given that the state in the first period is chosen according to α and that actions are chosen according to π.

There is a recursive definition of the value function of a policy π that is also a way to compute it when the horizon T is finite. This definition requires some concepts that we shall now introduce.

Given a belief state $\beta \in \Delta(S)$, a joint action $a \in A$ and a joint observation $o \in O$, let $\mathcal{T}(o|\beta, a)$ denote the probability that the agents receive joint observation o if they take joint action a in a period t in which the state is chosen according to β. This probability is defined as

$$\mathcal{T}(o|\beta, a) = \sum_{s \in S} \beta(s) \sum_{s' \in S} \mathbb{P}(s, a, s')\mathbb{G}(a, s', o) \tag{5}$$

Given a belief state $\beta \in \Delta(S)$, a joint action $a \in A$ and a joint observation $o \in O$, the **updated belief state** $\beta^{ao} \in \Delta(S)$ of β with respect to a and o is defined as (for each $s' \in S$),

$$\beta^{ao}(s') = \frac{\mathbb{G}(a,s',o)\left[\sum_{s \in S} \beta(s)\mathbb{P}(s,a,s')\right]}{\mathcal{T}(o|\beta,a)} \quad \text{if } \mathcal{T}(o|\beta, a) > 0 \tag{6}$$

$$\beta^{ao}(s') = 0 \quad \text{if } \mathcal{T}(o|\beta, a) = 0 \tag{7}$$

Given a belief state $\beta \in \Delta(S)$ and a joint action $a \in A$, $R(\beta, a)$ denotes $\sum_{s \in S} \beta(s)R(s, a)$. Using the above definitions and notations, the value $V(\alpha, \pi)$ of π is defined as follows:

$$V(\alpha, \pi) = V(\alpha, \pi, \varnothing) \tag{8}$$

where $V(\alpha, \pi, \varnothing)$ is defined by recursion using equations (9), (10) and (11), given below. These equations are a straight reformulation of the classical Bellman equations for finite horizon problems.

- For histories of null length

$$V(\alpha, \pi, \varnothing) = R(\alpha, \pi(\varnothing)) + \sum_{o \in O} \mathcal{T}(o|\alpha, \pi(\varnothing))V(\alpha^{\pi(\varnothing)o}, \pi, o) \tag{9}$$

  $\pi(\varnothing)$ denotes the joint action $\langle \pi_1(\varnothing), \pi_2(\varnothing), \cdots, \pi_n(\varnothing)\rangle$ and $\alpha^{\pi(\varnothing)o}$ denotes the updated state of α given $\pi(\varnothing)$ and the joint observation o.

- For non-terminal histories. For any $\alpha' \in \Delta(S)$, for each t of $\{1, \ldots, T-2\}$, for each tuple of sequences of t observations $o^{1:T} = \langle o_1^{1:T}, o_2^{1:T}, \cdots, o_n^{1:T} \rangle$ where $o_i^{1:T} \in \times_T O_i$ is a sequence of t observations of agent $i \in I$:

$$V(\alpha', \pi, o^{1:T}) \quad = \quad R(\alpha', \pi(o^{1:T})) + \sum_{o \in O} \mathcal{T}(o|\alpha', \pi(o^{1:T})) V(\alpha'^{\pi(o^{1:T})o}, \pi, o^{1:T}.o) \quad (10)$$

  $\alpha'^{\pi(o^{1:T})o}$ is the updated state of α' given the joint action $\pi(o^{1:T})$ and joint observation $o = \langle o_1, o_2, \cdots, o_n \rangle$ and $o^{1:T}.o$ is the tuple of sequences of $(t+1)$ observations $\langle o_1^{1:T}.o_1, o_2^{1:T}.o_2, \cdots, o_n^{1:T}.o_n \rangle$.

- For terminal histories. For any $\alpha' \in \Delta(S)$, for each tuple of sequences of $(T$ - $1)$ observations $o^{1:T-1} = \langle o_1^{1:T-1}, o_2^{1:T-1}, \cdots, o_n^{1:T-1} \rangle$:

$$V(\alpha', \pi, o^{1:T-1}) \quad = R(\alpha, \pi(o^{1:T-1})) \quad = \sum_{s \in S} \alpha'(s) R(s, \pi(o^{1:T-1})) \quad (11)$$

An **optimal policy** π^* is a policy with the best possible value, verifying:

$$V(\alpha, \pi^*) \geq V(\alpha, \pi) \qquad \forall \pi \in \Pi. \quad (12)$$

An important fact about DEC-POMDPs, based on the following theorem, is that we can restrict ourselves to the set of pure policies when looking for a solution to a DEC-POMDP.

Theorem 2.1. *A DEC-POMDP has at least one optimal pure joint policy.*

Proof: See proof in the work of Nair et al. (2003). □

2.6 Overview of DEC-POMDPs Solutions and Limitations

As detailed in the work of Oliehoek et al. (2008), existing methods for solving DEC-POMDPs with finite-horizon belong to several broad families: "brute force", alternating maximization, search algorithms and dynamic programming.

Brute Force The simplest approach for solving a DEC-POMDP is to enumerate all possible joint policies and to evaluate them in order to find the optimal one. However, such a method becomes quickly *untractable* as the number of joint policies is doubly exponential in the horizon of the problem.

Alternating Maximization Following Chadès, Scherrer, and Charpillet (2002) and Nair et al. (2003), one possible way to solve DEC-POMDPs is for each agent (or each small group of agents) to alternatively search for a better policy while all the other agents freeze their own policy. Called *alternating maximization* by Oliehoek and *alternated co-evolution* by Chadès this method guarantees only to find a Nash equilibria, that is a *locally optimal joint policy.*

Heuristic Search Algorithms The concept was introduced by Szer, Charpillet, and Zilberstein (2005) and relies on heuristic search for looking for an optimal joint policy, using an admissible approximation of the value of the optimal joint policy. As the search progresses, joint policies that will provably by worse that the current admissible solution are pruned. Szer et al. used underlying MDPs or POMDPs to compute the admissible heuristic, Oliehoek et al. (2008) introduced a better heuristic based on the resolution of a Bayesian Game with a carefully crafted cost function. Currently, Oliehoek's method called GMAA* (for Generic Multi-Agent A*) is the quickest exact method on a large set of benchmarks. But, as every exact method, it is limited to quite simple problems.

Dynamic Programming The work from Hansen, Bernstein, and Zilberstein (2004) adapts solutions designed for POMDPs to the domain of DEC-POMDPs. The general idea is to start with policies for 1 time step that are used to build 2 time step policies and so on. But the process is clearly less efficient that the heuristic search approach as an exponential number of policies must be constructed and evaluated at each iteration of the algorithm. Some of these policies can be pruned but, once again, pruning is less efficient.

As exposed in more details in the paper by Oliehoek et al. (2008), several others approaches have been developed for subclasses of DEC-POMDPs. For example, special settings where agents are allowed to communicate and exchange informations or settings where the transition function can be split into independant transition functions for each agent have been studied and found easier to solve than "generic" DEC-POMDPs.

3. Sequence-Form of DEC-POMDPs

This section introduces the fundamental concept of policies in "sequence-form". A new formulation of a DEC-POMDP is thus possible and this leads to a Non-Linear Program (NLP) the solution of which defines an optimal solution to the DEC-POMDP.

3.1 Policies in Sequence-Form

A **history function** p of an agent i is a mapping from the set of histories to the interval $[0, 1]$. The value $p(h)$ is the **weight** of the history h for the history function p. A policy π_i defines a probability function over the set of histories of the agent i by saying that, for each history h_i of $\mathcal{H}_i$, $p(h_i)$ is the conditional probability of h_i given an observation sequence $(o_i^0.o_i^1.\cdots.o_i^t)$ and π_i.

If every policy defines a policy function, not every policy function can be associated to a *valid* policy. Some constraints must be met. In fact, a history function p is a **sequence-form policy** for agent i when the following constraints are met:

$$\sum_{a \in A_i} p(a) = 1, \tag{13}$$

$$-p(h) + \sum_{a \in A_i} p(h.o.a) = 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i, \tag{14}$$

where $h.o.a$ denotes the history obtained on concatenating o and a to h. This definition appears in a slightly different form as Lemma 5.1 in the work of Koller et al. (1994).

Variables: $x(h)$, $\forall h \in \mathcal{H}_i$,

$$\sum_{a \in A_i} x(a) = 1 \tag{15}$$

$$-x(h) + \sum_{a \in A_i} x(h.o.a) = 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{16}$$

$$x(h) \geq 0, \quad \forall h \in \mathcal{H}_i \tag{17}$$

Table 1: **Policy Constraints**. This set of linear inequalities, once solved, provide a valid sequence-form policy for the agent i. That is, from the weights $x(h)$, it is possible to define a policy for the agent i.

A sequence-form policy can be stochastic as the probability of choosing action a in the information set $h.o$ is $p(h.o.a)/p(h)$. The **support** $S(p)$ of a sequence-form policy is made of the set of histories that have a non-negative weight, *i.e.* $S(p) = \{h \in \mathcal{H}_i \mid p(h) > 0\}$. As a sequence-form policy p defines a unique policy π for an agent, a sequence-form policy will be called a *policy* in the rest of this paper when no ambiguity is present.

The set of policies in the sequence-form of agent i shall be denoted by X_i. The set of pure policies in the sequence-form shall be denoted by $\hat{X}_i \subset X_i$.

In a way similar to the definitions of Section 2.4, we define a **sequence-form joint policy** as a tuple of sequence-form policies, one for each agent. The weight of the joint history $h = \langle h_i \rangle$ of a sequence-form joint policy $\langle p_1, p_2, \cdots, p_n \rangle$ is the product $\prod_{i \in I} p_i(h_i)$. The set of joint policies in the sequence-form $\times_{i \in I} X_i$ shall be denoted by X and the set of i-**reduced sequence-form joint policy** is called X_{-i}.

3.2 Policy Constraints

A policy of agent i in the sequence-form can be found by solving a set of linear inequalities (LI) found in Table 1. These LI merely implement the definition of a policy in the sequence-form. The LI contains one variable $x(h)$ for each history $h \in \mathcal{H}_i$ to represent the weight of h in the policy. A solution x^* to these LI constitutes a policy in the sequence-form.

Example In the Section E.1 of the Appendices, the policy constraints for the decentralized Tiger problem are given for 2 agents and a horizon of 2.

Notice that in the policy constraints of an agent, each variable is only constrained to be non-negative whereas by the definition of a policy in sequence-form, the weight of a history must be in the interval $[0, 1]$. Does it mean that a variable in a solution to the policy constraints can assume a value higher than 1? Actually, the policy constraints are such that they prevent any variable from assuming a value higher than 1 as the following lemma shows.

Lemma 3.1. *In every solution x^* to* (15)-(17)*, for each $h \in \mathcal{H}_i$, $x^*(h)$ belongs to the* $[0, 1]$ *interval.*

Proof: This can be shown by forward induction.

Every $x(h)$ being non-negative (see Eq. (17)), it is also the case for every action a of A_i. Then, no $x(a)$ can be greater than 1 otherwise constraint (15) would be violated. So, $\forall h \in \mathcal{H}_i^1$, (*i.e.* $\forall a \in A_i$), we have $x(h)$ belong to $[0, 1]$.

If every h of $\mathcal{H}_i^t$ is such that $x(h) \in [0, 1]$, the previous reasoning applied using constraint (16) leads evidently to the fact that $x(h) \in [0, 1]$ for every h of $\mathcal{H}_i^{t+1}$.

Thereby, by induction this holds for all t. □

Later in this article, in order to simplify the task of looking for joint policies, the policy constraints LI will be used to find pure policies. Looking for pure policies is not a limitation as finite-horizon DEC-POMDPs admit deterministic policies when the policies are defined on information set. In fact, pure policies are needed in two of the three MILPs we build in order to solve DEC-POMDPs, otherwise their derivation would not be possible (see Sections 4 and 5.4).

Looking for pure policies, an obvious solution would be to impose that every variable $x(h)$ belongs to the set $\{0, 1\}$. But, when solving a mixed integer linear program, it is generally a good idea to limit the number of integer variables as each integer variable is a possible node for the branch and bound method used to assign integer values to the variables. A more efficient implementation of a mixed integer linear program is to take advantage of the following lemma to impose that *only* the weights of the *terminal* histories take 0 or 1 as possible values.

Lemma 3.2. *If in* (15)-(17), (17) *is replaced by,*

$$x(h) \geq 0, \quad \forall h \in \mathcal{N}_i \tag{18}$$

$$x(h) \in \{0, 1\}, \quad \forall h \in \mathcal{E}_i \tag{19}$$

then in every solution x^* to the resulting LI, for each $h \in \mathcal{H}_i$, $x^*(h) = 0$ or 1. We will speak of a 0-1 LI.

Proof: We can prove this by backward induction. Let h be a history of length T - 1. Due to (16), for each $o \in O_i$, there holds,

$$x^*(h) = \sum_{a \in A_i} x^*(h.o.a). \tag{20}$$

Since h is a history of length T - 1, each history $h.o.a$ is a terminal history. Due to Lemma 3.1, $x^*(h) \in [0, 1]$. Therefore, the sum on the right hand side of the above equation is also in [0, 1]. But due to (19), each $x^*(h.o.a) \in \{0, 1\}$. Hence the sum on the right hand side is either 0 or 1, and not any value in between. Ergo, $x^*(h) \in \{0, 1\}$ and not any value in between. By this same reasoning, we can show that $x^*(h) \in \{0, 1\}$ for every non-terminal history h of length T - 2, T - 3, $\cdots$, 1. □

To formulate the linear inequalities of Table 1 in memory, we require space that is only exponential in the horizon. For each agent $i \in I$, the size of $\mathcal{H}_i$ is $\sum_{t=1}^{T} |A_i|^t |O_i|^{t-1}$. It is then exponential in T and the number of variables in the LP is also exponential in T. The number of constraints in the LI of Table 1 is $\sum_{t=0}^{T-1} |A_i|^t |O_i|^t$, meaning that the number of constraints of the LI is also exponential in T.

3.3 Sequence-Form of a DEC-POMDP

We are now able to give a formulation of a DEC-POMDP based on the use of sequence-form policies. We want to stress that this is *only* a re-formulation, but as such will provide us with new ways of solving DEC-POMDPs with mathematical programming.

Given a "classical" formulation of a DEC-POMDP (see Section 2.1), the equivalent **sequence-form DEC-POMDP** is a tuple $\langle I, \{\mathcal{H}_i\}, \Psi, \mathcal{R}\rangle$ where:

- $I = \{1, 2, \cdots, n\}$ is a set of agents.
- For each agent $i \in I$, $\mathcal{H}_i$ is the set of histories of length less than or equal to T for the agent i, as defined in the previous section. Each set $\mathcal{H}_i$ is derived using the sets A_i and O_i.
- Ψ is the joint history conditional probability function. For each joint history $j \in \mathcal{H}$, $\Psi(\alpha, j)$ is the probability of j occurring conditional on the agents taking joint actions according to it and given that the initial state of the DEC-POMDP is α. This function is derived using the set of states S, the state transition function $\mathbb{P}$ and the joint observation function $\mathbb{G}$.
- $\mathcal{R}$ is the joint history value. For each joint history $j \in \mathcal{H}$, $\mathcal{R}(\alpha, j)$ is the value of the expected reward the agents obtain if the joint history j occurs. This function is derived using the set of states S, the state transition function $\mathbb{P}$, the joint observation function $\mathbb{G}$ and the reward function R. Alternatively, $\mathcal{R}$ can be described as a function of Ψ and R.

This formulation folds S, $\mathbb{P}$ and $\mathbb{G}$ into Ψ and $\mathcal{R}$ by relying on the set of histories. We will now give more details about the computation of Ψ and $\mathcal{R}$.

$\Psi(\alpha, j)$ is the **conditional probability** that the sequence of joint observations received by the agents till period t is $(o^1(j).o^2(j).\cdots.\ o^{t-1}(j))$ **if** the sequence of joint actions taken by them till period t - 1 is $(a^1(j).\ a^2(j).\ \cdots.\ a^{t-1}(j))$ **and** the initial state of the DEC-POMDP is α. That is,

$$\Psi(\alpha, j) \quad = \quad \text{Prob.}(o^1(j).o^2(j).\cdots.o^{t-1}(j)|\alpha, a^1(j).a^2(j).\cdots.a^{t-1}(j)) \tag{21}$$

This probability is the product of the probabilities of seeing observation $o^k(j)$ given the appropriate belief state and action chosen at time k, that is:

$$\Psi(\alpha, j) \quad = \quad \prod_{k=1}^{t-1} \mathcal{T}(o^k(j)|\beta_j^{k-1}, a^k(j)) \tag{22}$$

where β_j^k is the probability distribution on S given that the agents have followed the joint history j up to time k, that is:

$$\beta_j^k(s) \quad = \quad \text{Prob.}(s|o^1(j).a^1(j).\cdots.o^k(j)). \tag{23}$$

Variables: $x_i(h)$, $\forall i \in I$, $\forall h \in \mathcal{H}_i$

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i(j_i) \tag{27}$$

subject to

$$\sum_{a \in A_i} x_i(a) = 1, \quad \forall i \in I \tag{28}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(h.o.a) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{29}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{30}$$

Table 2: **NLP**. This non-linear program expresses the constraints for finding a sequence-form joint policy that is an optimal solution to a DEC-POMDP.

Regarding the **value of a joint history**, it is defined by:

$$\mathcal{R}(\alpha, j) = \overline{R}(\alpha, j)\Psi(\alpha, j) \tag{24}$$

where

$$\overline{R}(\alpha, j) = \sum_{k=1}^{t} \sum_{s \in S} \beta_j^{k-1}(s) R(s, a^k(j)). \tag{25}$$

Thus, $\mathcal{V}(\alpha, p)$, the **value of a sequence-form joint policy** p, is the weighted sum of the value of the histories in its support:

$$\mathcal{V}(\alpha, p) = \sum_{j \in \mathcal{H}} p(j) \mathcal{R}(\alpha, j) \tag{26}$$

with $p(j) = \prod_{i \in I} p_i(j_i)$.

3.4 Non-Linear Program for Solving DEC-POMDPs.

By using the sequence-form formulation of a DEC-POMDP, we are able to express joint policies as sets of linear constraints and to assess the value of every policy. Solving a DEC-POMDP amounts to finding the policy with the maximal value, which can be done with the non-linear program (NLP) of Table 2 where, once again, the x_i variables are the weights of the histories for the agent i.

Example An example of the formulation of such an NLP can be found in the Appendices, in Section E.2. It is given for the decentralized Tiger problem with 2 agents and an horizon of 2.

The constraints of the program form a convex set, but the objective function is not concave (as explained in appendix A). In the general case, solving non-linear program is

very difficult and there are no generalized method that guarantee finding a global maximum point. However, this particular NLP is in fact a Multilinear Mathematical Program (see Drenick, 1992) and this kind of programs are still very difficult to solve. When only two agents are considered, one speaks of bilinear programs, that can be solved more easily (Petrik & Zilberstein, 2009; Horst & Tuy, 2003).

An evident, but inefficient, method to find a global maximum point is to evaluate *all* the extreme points of the set of feasible solutions of the program since it is known that every global as well as local maximum point of a non-concave function lies at an extreme point of such a set (Fletcher, 1987). This is an inefficient method because there is no test that tells if an extreme point is a local maximum point or a global maximum point. Hence, unless all extreme points are evaluated, we cannot be sure of having obtained a global maximum point. The set of feasible solutions to the NLP is X, the set of T-step joint policies. The set of extreme points of this set is $\hat{X}$, the set of pure T-step joint policies, whose number is doubly exponential in T and exponential in n. So enumerating the extreme points for this NLP is untractable.

Our approach, developed in the next sections, is to linearize the objective function of this NLP in order to deal only with linear programs. We will describe two ways for doing this: one is based on combinatorial consideration (Section 4) and the other is based on game theory concepts (Section 5). In both cases, this shall mean adding more variables and constraints to the NLP, but upon doing so, we shall derive *mixed integer linear programs* for which it is possible to find a global maximum point and hence an optimal joint policy of the DEC-POMDP.

4. From Combinatorial Considerations to Mathematical Programming

This section explains how it is possible to use combinatorial properties of DEC-POMDPs to transform the previous NLP into a mixed integer linear program. As shown, this mathematical program belongs to the family of 0-1 Mixed Integer Linear Programs, meaning that some variables of this linear program must take integer values in the set $\{0, 1\}$.

4.1 Linearization of the Objective Function

Borrowing ideas from the field of *Quadratic Assignment Problems* (Papadimitriou & Steiglitz, 1982), we turn the non-linear objective function of the previous NLP into a linear objective function and linear constraints involving new variables z that must take integer values. The variable $z(j)$ represents the product of the $x_i(j_i)$ variables.

Thus, the objective function that was:

$$\text{maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) \prod_{i \in I} x_i(j_i) \tag{31}$$

can now be rewritten as

$$\text{maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{32}$$

where $j = \langle j_1, j_2, \cdots, j_n \rangle$.

We must ensure that there is a two way mapping between the value of the new variables z and the x variables for any solution of the mathematical program, that is:

$$z^*(j) \quad = \quad \prod_{i \in I} x_i^*(j_i). \tag{33}$$

For this, we will restrict ourself to *pure policies* where the x variables can only be 0 or 1. In that case, the previous constraint (33) becomes:

$$z^*(j) = 1 \quad \Leftrightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I \tag{34}$$

There, we take advantage on the fact that the support of a pure policy for an agent i is composed of $|O_i|^{T-1}$ terminal histories to express these new constraints. On the one hand, to guarantee that $z(j)$ is equal to 1 only when enough x variables are also equal to 1, we write:

$$\sum_{i=1}^{n} x_i(j_i) - nz(j) \quad \geq \quad 0, \quad \forall j \in \mathcal{E}. \tag{35}$$

On the other hand, to limit the number of $z(j)$ variables that can take a value of 1, we will enumerate the number of joint terminal histories to end up with:

$$\sum_{j \in \mathcal{E}} z(j) \quad = \quad \prod_{i \in I} |O_i|^{T-1}. \tag{36}$$

The constraints (35) would weight heavily on any mathematical program as there would be one constraint for each terminal *joint* history, a number which is exponential in n and T. Our idea to reduce this number of constraints is not to reason about *joint histories* but with *individual histories*. An history h of agent i is part of the support of the solution of the problem (*i.e.*, $x_i(h) = 1$) if and only if the number of joint histories it belongs to $(\sum_{j' \in \mathcal{E}_{-i}} z(\langle h, j' \rangle))$ is $\prod_{k \in I \setminus \{i\}} |O_k|^{T-1}$. Then, we suggest to replace the $\prod |\mathcal{E}_i|$ constraints (35)

$$\sum_{i=1}^{n} x_i(j_i) - nz(j) \geq 0, \quad \forall j \in \mathcal{E}. \tag{35}$$

by the $\sum |\mathcal{E}_i|$ constraints

$$\begin{aligned} \sum_{j' \in \mathcal{E}_{-i}} z(\langle h, j' \rangle) \quad &= \quad \frac{\prod_{k \in I} |O_k|^{T-1}}{|O_i|^{T-1}} x_i(h) \\ &= \quad x_i(h) \prod_{k \in I \setminus \{i\}} |O_k|^{T-1}, \quad \forall i \in I, \forall h \in \mathcal{E}_i. \end{aligned} \tag{37}$$

4.2 Fewer Integer Variables

The linearization of the objective function rely on the fact that we are dealing with pure policies, meaning that every x and z variable is supposed to value either 0 or 1. As solving linear programs with integer variables is usually based on the "branch and bound" technique

Variables:
$x_i(h)$, $\forall i \in I$, $\forall h \in \mathcal{H}_i$
$z(j)$, $\forall j \in \mathcal{E}$

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \tag{38}$$

subject to:

$$\sum_{a \in A_i} x_i(a) = 1, \quad \forall i \in I \tag{39}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(h.o.a) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{40}$$

$$\sum_{j' \in H^T_{-i}} z(\langle h, j' \rangle) = x_i(h) \prod_{k \in I \setminus \{i\}} |O_k|^{T-1}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{41}$$

$$\sum_{j \in \mathcal{E}} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{42}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{43}$$

$$x_i(h) \in \{0, 1\}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{44}$$

$$z(j) \in [0, 1], \quad \forall j \in \mathcal{E} \tag{45}$$

Table 3: **MILP**. This 0-1 mixed integer linear program finds a sequence-form joint policy that is an optimal solution to a DEC-POMDP.

(Fletcher, 1987), for efficiency reasons, it is important to reduce the number of integer variables in our mathematical programs.

As done in Section 3.2, we can relax most x variables and allow them to take non-negative values provided that the x values for terminal histories are constrained to integer values. Furthermore, as proved by the following lemma, these constraints on x *also* guarantee that z variables only take their value in $\{0, 1\}$.

We eventually end up with the following linear program with real and integer variables, thus called an **0-1 mixed integer linear program** (MILP). The MILP is shown in Table 3.

Example In Section E.3 of the Appendices, an example if such MILP is given for the problem of the decentralized Tiger for 2 agents and an horizon of 2.

Lemma 4.1. *In every solution (x^*, z^*) to the* **MILP** *of Table 3, for each $j \in \mathcal{E}$, $z^*(j)$ is either 0 or 1.*

Proof: Let (x^*, z^*) be a solution of **MILP**. Let,

$$S(z) = \{j \in \mathcal{E} | z^*(j) > 0\} \tag{46}$$

$$S_i(x_i) = \{h \in \mathcal{E}_i | x_i^*(h) = 1\}, \quad \forall i \in I \tag{47}$$

$$S_i(z, j') = \{j \in \mathcal{E} | j_{-i} = j', z^*(j) > 0\}, \quad \forall i \in I, \forall j' \in \mathcal{E}_{-i} \tag{48}$$

Now, due to (42) and (45), $|S(z)| \geq \prod_{i \in I} |O_i|^{T-1}$. By showing that $|S(z)| \leq \prod_{i \in I} |O_i|^{T-1}$, we shall establish that $|S(z)| = \prod_{i \in I} |O_i|^{T-1}$. Then due to the upper bound of 1 on each z variable, the implication will be that $z^*(j)$ is 0 or 1 for each terminal joint history j thus proving the statement of the lemma.

Note that by Lemma (3.2), for each agent i, x_i^* is a pure policy. Therefore, we have that $|S_i(x)| = |O_i|^{T-1}$. This means that in the set of constraints (41), an i-reduced terminal joint history $j' \in \mathcal{E}_{-i}$ will appear on the right hand side not more than $|O_i|^{T-1}$ times when in the left hand side, we have $x_i^*(h) = 1$. Thus, $\forall j' \in \mathcal{E}_{-i}$,

$$|S_i(z, j')| \quad \leq \quad |O_i|^{T-1}. \tag{49}$$

Now, we know that for each agent i and for each history $h \in \mathcal{H}_i$, $x_i^*(h)$ is either 0 or 1 since x_i^* is a pure policy. So, given an i-reduced terminal joint history j', $\prod_{k \in I \setminus \{i\}} x_k^*(j'_k)$ is either 0 or 1. Secondly, due to (41), the following implication clearly holds for each terminal joint history j,

$$z^*(j) > 0 \quad \Rightarrow \quad x_i^*(j_i) = 1, \quad \forall i \in I. \tag{50}$$

Therefore, we obtain

$$\begin{aligned} |S_i(z, j')| \quad &\leq \quad |O_i|^{T-1} && (51) \\ &= \quad |O_i|^{T-1} \prod_{k \in I \setminus \{i\}} x_k^*(j'_k). && (52) \end{aligned}$$

As a consequence,

$$\begin{aligned} \sum_{j' \in \mathcal{E}_{-i}} |S_i(z, j')| \quad &\leq \quad \sum_{j' \in \mathcal{E}_{-i}} |O_i|^{T-1} \prod_{k \in I \setminus \{i\}} x_k^*(j'_k) && (53) \\ &= \quad |O_i|^{T-1} \sum_{j' \in \mathcal{E}_{-i}} \prod_{k \in I \setminus \{i\}} x_k^*(j'_k) && (54) \\ &= \quad |O_i|^{T-1} \prod_{k \in I \setminus \{i\}} \sum_{h' \in \mathcal{E}_k} x_k^*(h') && (55) \\ &= \quad |O_i|^{T-1} \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} && (56) \\ &= \quad \prod_{j \in I} |O_j|^{T-1}. && (57) \end{aligned}$$

Since $\bigcup_{j' \in \mathcal{E}_{-i}} S_i(z, j') = S(z)$, there holds that $\sum_{j' \in \mathcal{E}_{-i}} |S_i(z, j')| = |S(z)|$. Hence,

$$|S(z)| \quad \leq \quad \prod_{j \in I} |O_j|^{T-1}. \tag{58}$$

Thus the statement of the lemma is proved. □

4.3 Summary

By using combinatorial considerations, it is possible to design a 0-1 MILP for solving a given DEC-POMDP. As proved by theorem 4.1, the solution of this MILP defines an optimal joint policy for the DEC-POMDP. Nevertheless, this MILP is quite large, with $O(k^T)$ constraints and $\sum_i |\mathcal{H}_i| + \prod_i |\mathcal{E}_i| = O(k^{nT})$ variables, $O(k^T)$ of these variables must take integer values. The next section details another method for the linearization of NLP which leads to a "smaller" mathematical program for the 2-agent case.

Theorem 4.1. *Given a solution (x^*, z^*) to* **MILP**, $x^* = \langle x_1^*, x_2^*, \cdots, x_n^* \rangle$ *is a pure T-period optimal joint policy in sequence-form.*

Proof: Due to the policy constraints and the domain constraints of each agent, each x_i^* is a pure sequence-form policy of agent i. Due to the constraints (41)-(42), each z^* values 1 if and only if the product $\prod_{i \in I} x_i(j_i)$ values 1. Then, by maximizing the objective function we are effectively maximizing the value of the sequence-form policy $\langle x_1^*, x_2^*, \cdots, x_n^* \rangle$. Thus, $\langle x_1^*, x_2^*, \cdots, x_n^* \rangle$ is an optimal joint policy of the original DEC-POMDP. □

5. From Game-Theoretical Considerations to Mathematical Programming

This section borrows concepts like "Nash equilibrium" and "regret" from game theory in order to design yet another 0-1 Mixed Integer Linear Program for solving DEC-POMDPs. In fact, two MILPs are designed, one that can only be applied for 2 agents and the other one for any number of agents. The main objective of this part is to derive a smaller mathematical program for the 2 agent case. Indeed, **MILP-2 agents** (see Table 4) has slightly less variables and constraints than **MILP** (see Table 3) and thus might prove easier to solve. On the other hand, when more than 2 agents are considered, the new derivation leads to a MILP which is only given for completeness as it is bigger than **MILP**.

Links between the fields of multiagent systems and game theory are numerous in the literature (see, for example, Sandholm, 1999; Parsons & Wooldridge, 2002). We will elaborate on the fact that the optimal policy of a DEC-POMDP is a Nash Equilibrium. It is in fact the Nash Equilibrium with the highest utility as the agents all share the same reward.

For the 2-agent case, the derivation we make in order to build the MILP is similar to the first derivation of Sandholm, Gilpin, and Conitzer (2005). We give more details of this derivation and adapt it to DEC-POMDP by adding an objective function to it. For more than 2 agents, our derivation can still be use to find Nash equilibriae with pure strategies.

For the rest of this article, we will make no distinction between a policy, a sequence-form policy or a strategy of an agent as, in our context, these concepts are equivalent. Borrowing from game theory, a joint policy will be denoted p or q, an individual policy p_i or q_i and a i-reduced policy p_{-i} or q_{-i}.

5.1 Nash Equilibrium

A Nash Equilibrium is a joint policy in which each policy is a best response to the reduced joint policy formed by the other policies of the joint policy. In the context of a sequence-form

DEC-POMDP, a policy $p_i \in X_i$ of agent i is said to be a **best response** to an i-reduced joint policy $q_{-i} \in X_{-i}$ if there holds that

$$\mathcal{V}(\alpha, \langle p_i, q_{-i}\rangle) - \mathcal{V}(\alpha, \langle p'_i, q_{-i}\rangle) \quad \geq \quad 0, \quad \forall p'_i \in X_i. \tag{59}$$

A joint policy $p \in X$ is a **Nash Equilibrium** if there holds that

$$\mathcal{V}(\alpha, p) - \mathcal{V}(\alpha, \langle p'_i, p_{-i}\rangle) \quad \geq \quad 0, \quad \forall i \in I, \forall p'_i \in X_i. \tag{60}$$

That is,

$$\sum_{h \in \mathcal{E}_i} \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h, j'\rangle) \prod_{k \in I \setminus \{i\}} p_k(j'_k)\big\{p_i(h) - p'_i(h)\big\} \quad \geq \quad 0, \quad \forall i \in I, \forall p'_i \in X_i. \tag{61}$$

The derivation of the necessary conditions for a Nash equilibrium consists of deriving the necessary conditions for a policy to be a best response to a reduced joint policy. The following program finds a policy for an agent i that is a best response to an i-reduced joint policy $q_{-i} \in X_{-i}$. Constraints (63)-(64) ensure that the policy defines a valid joint policy (see Section 3.2) and the objective function is a traduction of the concept of best response.

Variables: $x_i(h)$, $\forall i \in I$, $\forall h \in \mathcal{H}_i$

$$\text{Maximize} \quad \sum_{h \in \mathcal{E}_i} \left\{ \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h, j'\rangle) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \right\} x_i(h) \tag{62}$$

subject to:

$$\sum_{a \in A_i} x_i(a) \quad = \quad 1 \tag{63}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(h.o.a) \quad = \quad 0, \quad \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{64}$$

$$x_i(h) \quad \geq \quad 0, \quad \forall h \in \mathcal{H}_i. \tag{65}$$

This linear program (LP) must still be refined so that its solution is not only a best response for agent i but a "global" best response, *i.e.*, the policy of *each* agent is a best response to all the other agents. This will mean introducing new variables (a set of variable for each agent). The main point will be to adapt the objective function as the current objective function, when applied to find "global" best response, would lead to a non-linear objective function where product of weights of policies would appear. To do this, we will make use of the *dual* of the program (LP).

The linear program (LP) has one variable $x_i(h)$ for each history $h \in \mathcal{H}_i$ representing the weight of h. It has one constraint per information set of agent i. In other words, each constraint of the linear program (LP) is uniquely labeled by an information set. For instance, the constraint (63) is labeled by the null information set $\varnothing$, and for each nonterminal history h and for each observation o, the corresponding constraint in (64) is labeled by the information set $h.o$. Thus, (LP) has n_i variables and m_i constraints.

As described in the appendix (see appendix B), the dual of (LP) is expressed as:

Variables: $y_i(\varphi)$, $\forall \varphi \in \Phi_i$

$$\text{Minimize} \quad y_i(\varnothing) \tag{66}$$

subject to:

$$y_i(\varphi(h)) - \sum_{o \in O_i} y_i(h.o) \;\geq\; 0, \quad \forall h \in \mathcal{N}_i \tag{67}$$

$$y_i(\varphi(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h, j' \rangle) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \;\geq\; 0, \quad \forall h \in \mathcal{E}_i \tag{68}$$

$$y_i(\varphi) \;\in\; (-\infty, +\infty), \quad \forall \varphi \in \Phi_i \tag{69}$$

where $\varphi(h)$ denotes the information set to which h belongs. The dual has one free variable $y_i(\dot{})$ for every *information set* of agent i. This is why the function $\varphi(h)$ (defined in Section 2.3) appears as a mapping from histories to information sets[1]. The dual program has one constraint per history of the agent. Thus, the dual has m_i variables and n_i constraints. Note that the objective of the dual is to minimize only $y_i(\varnothing)$ because in the primal (LP), the right hand side of all the constraints, except the very first one, is a 0.

The theorem of duality (see the appendix B), applied to the primal (LP) (62)-(65) and the transformed dual (66)-(69), says that their solutions have the same value. Mathematically, that means that:

$$\sum_{h \in \mathcal{E}_i} \left\{ \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h, j' \rangle) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \right\} x_i^*(h) \;=\; y_i^*(\varnothing). \tag{70}$$

Thus, the value of the joint policy $\langle x_i^*, q_{-i} \rangle$ can be expressed either as

$$\mathcal{V}(\alpha, \langle x_i^*, q_{-i} \rangle) \;=\; \sum_{h \in \mathcal{E}_i} \Big\{ \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h, j' \rangle) \prod_{k \in I \setminus \{i\}} q_k(j'_k) \Big\} x_i^*(h) \tag{71}$$

or as

$$\mathcal{V}(\alpha, \langle x_i^*, q_{-i} \rangle) \;=\; y_i^*(\varnothing). \tag{72}$$

Due to the constraints (63) and (64) of the primal LP, there holds that

$$y_i^*(\varnothing) \;=\; y_i^*(\varnothing) \Big\{ \sum_{a \in A_i} x_i^*(a) \Big\} + \sum_{h \in \mathcal{N}_i} \sum_{o \in O_i} y_i^*(h.o) \Big\{ -x_i^*(h) + \sum_{a \in A_i} x_i^*(h.o.a) \Big\} \tag{73}$$

as constraint (63) guarantees that the first term in the braces is 1 and constraints (65) guarantee that each of the remaining terms inside the braces is 0. The right hand side of (73) can be rewritten as

$$\begin{aligned} \sum_{a \in A_i} x_i^*(a) \big\{ y_i^*(\varnothing) \quad & - \textstyle\sum_{o \in O_i} y_i^*(a.o) \big\} \quad + \sum_{h \in \mathcal{N}_i \setminus A_i} x_i^*(h) \big\{ y_i^*(\varphi(h)) - \sum_{o \in O_i} y_i^*(h.o) \big\} \\ & \qquad + \sum_{h \in \mathcal{E}_i} x_i^*(h) y_i^*(\varphi(h)) \\ = \textstyle\sum_{h \in \mathcal{N}_i} x_i^*(h) \big\{ y_i^*(\varphi(h)) - & \sum_{o \in O_i} y_i^*(h.o) \big\} + \sum_{h \in \mathcal{E}_i} x_i^*(h) y_i^*(\varphi(h)) \end{aligned} \tag{74}$$

1. As $h.o$ is an information set, $y_i(h.o)$ is a shortcut in writing for $y_i(\varphi(h.o))$.

So, combining equations (70) and (74), we get

$$\sum_{h\in\mathcal{N}_i} x_i^*(h) \qquad \big\{y_i^*(\varphi(h)) - \sum_{o\in O_i} y_i^*(h.o)\big\} \\ + \sum_{h\in\mathcal{E}_i} x_i^*(h)\big\{y_i^*(\varphi(h)) - \sum_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,\langle h,j'\rangle) \prod_{k\in I\setminus\{i\}} q_k(j'_k)\big\} = 0 \qquad (75)$$

It is time to introduce supplementary variables w for each information set. These variables, usually called *slack* variables, are defined as:

$$y_i(\varphi(h)) - \sum_{o\in O_i} y_i(h.o) = w_i(h), \quad \forall h \in \mathcal{N}_i \qquad (76)$$

$$y_i(\varphi(h)) - \sum_{j'\in\mathcal{E}_{-i}} \mathcal{R}(\alpha,\langle h,j'\rangle) \prod_{k\in I\setminus\{i\}} q_k(j'_k) = w_i(h), \quad \forall h \in \mathcal{E}_i. \qquad (77)$$

As shown is Section C of the appendix, these slack variables correspond to the concept of regret as defined in game theory. The regret of an history expresses the loss in accumulated reward the agent incurs when he acts according to this history rather than according to a history which would belong to the optimal joint policy.

Thanks to the slack variables, we can furthermore rewrite (75) as simply

$$\sum_{h\in\mathcal{N}_i} x_i^*(h)w_i^*(h) + \sum_{h\in\mathcal{E}_i} x_i^*(h)w_i^*(h) = 0 \qquad (78)$$

$$\sum_{h\in\mathcal{H}_i} x_i^*(h)w_i^*(h) = 0. \qquad (79)$$

Now, (79) is a sum of n_i products, n_i being the size of $\mathcal{H}_i$. Each product in this sum is necessarily 0 because both $x_i(h)$ and $w_i(h)$ are constrained to be nonnegative in the primal and the dual respectively. This property is strongly linked to the complementary slackness optimality criterion in linear programs (see, for example, Vanderbei, 2008). Hence, (79) is equivalent to

$$x_i^*(h)w_i^*(h) = 0, \quad \forall h \in \mathcal{H}_i. \qquad (80)$$

Back to the framework of DEC-POMDPs, these constraints are written:

$$p_i(h)\mu_i(\langle h, q_{-i}\rangle) = 0, \quad \forall h \in \mathcal{H}_i. \qquad (81)$$

To sum up, solving the following mathematical program would give an optimal joint policy for the DEC-POMDP. But constraints (87) are non-linear and thus prevent us from solving this program directly. The linearization of these constraints, called complementarity constraints, is the subject of the next section.

Variables:
$x_i(h)$, $w_i(h)$ $\forall i \in I$ and $\forall h \in \mathcal{H}_i$
$y_i(\varphi)$ $\forall i \in I$ and $\forall \varphi \in \Phi_i$

$$\text{Maximize} \quad y_1(\varnothing) \qquad (82)$$

subject to:

$$\sum_{a \in A_i} x_i(a) = 1 \tag{83}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(h.o.a) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{84}$$

$$y_i(\varphi(h)) - \sum_{o \in O_i} y_i(h.o) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{85}$$

$$y_i(\varphi(h)) - \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h, j' \rangle) \prod_{k \in I \setminus \{i\}} x_k(j'_k) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{86}$$

$$x_i(h) w_i(h) = 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{87}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{88}$$

$$w_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{89}$$

$$y_i(\varphi) \in (-\infty, +\infty), \quad \forall i \in I, \forall \varphi \in \Phi_i \tag{90}$$

5.2 Dealing with Complementarity Constraints

This section explains how the non-linear constraints $x_i(h)w_i(h) = 0$ in the previous mathematical program can be turned into sets of linear constraints and thus lead to a mixed integer linear programming formulation of the solution of a DEC-POMDP.

Consider a complementarity constraint $ab = 0$ in variables a and b. Assume that the lower bound on the values of a and b is 0. Let the upper bounds on the values of a and b be respectively u_a and u_b. Now let c be a 0-1 variable. Then, the complementarity constraint $ab = 0$ can be separated into the following equivalent pair of linear constraints,

$$a \leq u_a c \tag{91}$$

$$b \leq u_b(1 - c). \tag{92}$$

In other words, if this pair of constraints is satisfied, then it is surely the case that $ab = 0$. This is easily verified. c can either be 0 or 1. If $c = 0$, then a will be set to 0 because a is constrained to be no more than $u_a c$ (and not less than 0); if $c = 1$, then b will be set to 0 since b is constrained to be not more than $u_b(1 - c)$ (and not less than 0). In either case, $ab = 0$.

Now consider each complementarity constraint $x_i(h)w_i(h) = 0$ from the non-linear program (82)-(90) above. We wish to separate each constraint into a pair of linear constraints. We recall that $x_i(h)$ represents the weight of h and $w_i(h)$ represents the regret of h. The first requirement to convert this constraint to a pair of linear constraints is that the lower bound on the values of the two terms be 0. This is indeed the case since $x_i(h)$ and $w_i(h)$ are both constrained to be non-negative in the NLP. Next, we require upper bounds on the weights of histories and regrets of histories. We have shown in Lemma 3.1 that the upper bound on the value of $x_i(h)$ for each h is 1. For the upper bounds on the regrets of histories, we require some calculus.

In any policy p_i of agent i there holds that

$$\sum_{h \in \mathcal{E}_i} p_i(h) \quad = \quad |O_i|^{T-1}. \tag{93}$$

Therefore, in every i-reduced joint policy $\langle q_1, q_2, \cdots, q_n \rangle \in X_{-i}$, there holds

$$\sum_{j' \in \mathcal{E}_{-i}} \prod_{k \in I \setminus \{i\}} q_k(j'_k) \quad = \quad \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} \tag{94}$$

Since the regret of a terminal history h of agent i given $\langle q_1, q_2, \cdots, q_n \rangle$ is defined as

$$\mu_i(h, q) \quad = \quad \max_{h' \in \varphi(h)} \sum_{j' \in \mathcal{E}_{-i}} \prod_{k \in I \setminus \{i\}} q_k(j'_k) \left\{ \mathcal{R}(\alpha, \langle h', j' \rangle) - \mathcal{R}(\alpha, \langle h, j' \rangle) \right\}, \tag{95}$$

we can conclude that an **upper bound** $\mathcal{U}_i(h)$ on the regret of a **terminal history** $h \in \mathcal{E}_i$ of agent i is,

$$\mathcal{U}_i(h) \quad = \quad \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} \left\{ \max_{h' \in \varphi(h)} \max_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h', j' \rangle) - \min_{j'' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h, j'' \rangle) \right\}. \tag{96}$$

Now let us consider the upper bounds on the regrets of non-terminal histories. Let φ be an information set of length t of agent i. Let $\mathcal{E}_i(\varphi) \subseteq \mathcal{E}_i$ denote the set of terminal histories of agent i such the first $2t$ elements of each history in the set are identical to φ. Let h be a history of length $t \leq T$ of agent i. Let $\mathcal{E}_i(h) \subseteq \mathcal{E}_i$ denote the set of terminal histories such that the first $2t$ - 1 elements of each history in the set are identical to h. Since in any policy p_i of agent i, there holds

$$\sum_{h' \in \mathcal{E}_i(h)} p_i(h') \leq |O_i|^{T-t} \tag{97}$$

we can conclude that an **upper bound** $\mathcal{U}_i(h)$ on the regret of a **nonterminal history** $h \in \mathcal{N}_i$ of length t agent i is

$$\mathcal{U}_i(h) = L_i \left\{ \max_{h' \in \mathcal{E}_i(\varphi(h))} \max_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h', j' \rangle) - \min_{g \in \mathcal{E}_i(h)} \min_{j'' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle g, j'' \rangle) \right\} \tag{98}$$

where

$$L_i \quad = \quad |O_i|^{T-t} \prod_{k \in I \setminus \{i\}} |O_k|^{T-1}. \tag{99}$$

Notice that if $t = T$ (that is, h is terminal) (98) reduces to (96).

So, the complementarity constraint $x_i(h) w_i(h) = 0$ can be separated into a pair of linear constraints by using a 0-1 variable $b_i(h)$ as follows,

$$x_i(h) \quad \leq \quad 1 - b_i(h) \tag{100}$$

$$w_i \quad \leq \quad \mathcal{U}_i(h) b_i(h) \tag{101}$$

$$b_i(h) \quad \in \quad \{0, 1\} \tag{102}$$

Variables:
$x_i(h)$, $w_i(h)$ and $b_i(h)$ for $i \in \{1,2\}$ and $\forall h \in \mathcal{H}_i$
$y_i(\varphi)$ for $i \in \{1,2\}$ and $\forall \varphi \in \Phi_i$

$$\text{Maximize} \quad y_1(\varnothing) \tag{103}$$

subject to:

$$\sum_{a \in A_i} x_i(a) = 1 \tag{104}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(h.o.a) = 0, \quad i = 1, 2, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{105}$$

$$y_i(\varphi(h)) - \sum_{o \in O_i} y_i(h.o) = w_i(h), \quad i = 1, 2, \forall h \in \mathcal{N}_i \tag{106}$$

$$y_1(\varphi(h)) - \sum_{h' \in \mathcal{E}_2} \mathcal{R}(\alpha, \langle h, h' \rangle) x_2(h') = w_1(h), \quad \forall h \in \mathcal{E}_1 \tag{107}$$

$$y_2(\varphi(h)) - \sum_{h' \in \mathcal{E}_1} \mathcal{R}(\alpha, \langle h', h \rangle) x_1(h') = w_2(h), \quad \forall h \in \mathcal{E}_2 \tag{108}$$

$$x_i(h) \leq 1 - b_i(h), \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{109}$$

$$w_i(h) \leq \mathcal{U}_i(h) b_i(h), \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{110}$$

$$x_i(h) \geq 0, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{111}$$

$$w_i(h) \geq 0, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{112}$$

$$b_i(h) \in \{0,1\}, \quad i = 1, 2, \forall h \in \mathcal{H}_i \tag{113}$$

$$y_i(\varphi) \in (-\infty, +\infty), \quad i = 1, 2, \forall \varphi \in \Phi_i \tag{114}$$

Table 4: **MILP-2 agents**. This 0-1 mixed integer linear program, derived from game theoretic considerations, finds optimal *stochastic* joint policies for DEC-POMDPs with 2 agents.

5.3 Program for 2 Agents

When we combine the policy constraints (Section 3.2), the constraints we have just seen for a policy to be a best response (Sections 5.1, 5.2) and a maximization of the value of the joint policy, we can derive a 0-1 mixed integer linear program the solution of which is an optimal joint policy for a DEC-POMDP for 2 agents. Table 4 details this program that we will call **MILP-2 agents**.

Example The formulation of the decentralized Tiger problem for 2 agents and for an horizon of 2 can be found in the appendices, in Section E.4

The variables of the program are the vectors x_i, w_i, b_i and y_i for each agent i. Note that for each agent $i \in I$ and for each history h of agent i, $\mathcal{U}_i(h)$ denotes the upper bound on the regret of history h.

A solution (x^*, y^*, w^*, b^*) to **MILP-2 agents** consists of the following quantities: (i) an optimal joint policy $x^* = \langle x_1^*, x_2^* \rangle$ which may be stochastic; (ii) for each agent $i = 1$, 2, for each history $h \in \mathcal{H}_i$, $w_i^*(h)$, the regret of h given the policy x_{-i}^* of the other agent; (iii) for each agent $i = 1$, 2, for each information set $\varphi \in \Phi_i$, $y_i^*(\varphi)$, the value of φ given the policy x_{-i} of the other agent; (iv) for each agent $i = 1$, 2, the vector b_i^* simply tells us which histories are not in the support of x_i^*; each history h of agent i such that $b_i^*(h) = 1$ is *not* in the support of x_i^*. Note that we can replace $y_1(\varnothing)$ by $y_2(\varnothing)$ in the objective function without affecting the program. We have the following result.

Theorem 5.1. *Given a solution (x^*, w^*, y^*, b^*) to* **MILP-2 agents**, $x^* = \langle x_1^*, x_2^* \rangle$ *is an optimal joint policy in sequence-form.*

Proof: Due to the policy constraints of each agent, each x_i^* is a sequence-form policy of agent i. Due to the constraints (106)-(108), y_i^* contains the values of the information sets of agent i given x_{-i}^*. Due to the complementarity constraints (109)-(110), each x_i^* is a best response to x_{-i}^*. Thus $\langle x_1^*, x_2^* \rangle$ is a Nash equilibrium. Finally, by maximizing the value of the null information set of agent 1, we are effectively maximizing the value of $\langle x_1^*, x_2^* \rangle$. Thus $\langle x_1^*, x_2^* \rangle$ is an optimal joint policy. □

In comparison with the **MILP** presented before in Table 3, **MILP-2 agents** should constitutes a particularly effective program in term of computation time for finding a 2-agent optimal T-period joint policy because it is a much smaller program. While the number of variables required by **MILP** is exponential in T and in n, the number of variables required by **MILP-2 agents** is exponential only in T. This represents a major reduction in size that should lead to an improvement in term of computation time.

5.4 Program for 3 or More Agents

When the number of agents is more than 2, the constraint (86) of the non-linear program (82)-(90) is no longer a complementarity constraint between 2 variables that could be linearized as before. In particular, the term $\prod_{k \in I \setminus \{i\}} x_k(j'_k)$ of the constraint (86) involves as many variables as there are different agents. To linearize this term, we will restrict ourselves once again to *pure* joint policies and exploit some combinatorial facts on the number of histories involved. This leads to the 0-1 mixed linear program called **MILP-n agents** and depicted in Table 5.

The variables of the program **MILP-n agents** are the vectors x_i, w_i, b_i and y_i for each agent i and the vector z. We have the following result.

Theorem 5.2. *Given a solution (x^*, w^*, y^*, b^*, z^*) to* **MILP-n agents**, $x^* = \langle x_1^*, x_2^*, \cdots, x_n^* \rangle$ *is a pure T-period optimal joint policy in sequence-form.*

Proof: Due to the policy constraints and the domain constraints of each agent, each x_i^* is a pure sequence-form policy of agent i. Due to the constraints (118)-(119), each y_i^* contains the values of the information sets of agent i given x_{-i}^*. Due to the complementarity constraints (122)-(123), each x_i^* is a best response to x_{-i}^*. Thus x^* is a Nash equilibrium. Finally, by maximizing the value of the null information set of agent 1, we are effectively maximizing the value of x^*. Thus x^* is an optimal joint policy. □

Variables:
$x_i(h)$, $w_i(h)$ and $b_i(h)$ $\forall i \in I$ and $\forall h \in \mathcal{H}_i$
$y_i(\varphi)$ $\forall i \in I$, $\forall \varphi \in \Phi_i$
$z(j)$ $\forall j \in \mathcal{E}$

$$\text{Maximize} \quad y_1(\varnothing) \tag{115}$$

subject to:

$$\sum_{a \in A_i} x_i(a) = 1 \tag{116}$$

$$-x_i(h) + \sum_{a \in A_i} x_i(h.o.a) = 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i, \forall o \in O_i \tag{117}$$

$$y_i(\varphi(h)) - \sum_{o \in O_i} y_i(h.o) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{118}$$

$$y_i(\varphi(h)) - \frac{1}{|O_i|^{T-1}} \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, \langle h, j_{-i} \rangle) z(j) = w_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{119}$$

$$\sum_{j' \in \mathcal{E}_{-i}} z(\langle h, j' \rangle) = x_i(h) \prod_{k \in I \setminus \{i\}} |O_k|^{T-1}, \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{120}$$

$$\sum_{j \in \mathcal{E}} z(j) = \prod_{i \in I} |O_i|^{T-1} \tag{121}$$

$$x_i(h) \leq 1 - b_i(h), \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{122}$$

$$w_i(h) \leq \mathcal{U}_i(h) b_i(h), \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{123}$$

$$x_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{N}_i \tag{124}$$

$$x_i(h) \in \{0, 1\} \quad \forall i \in I, \forall h \in \mathcal{E}_i \tag{125}$$

$$w_i(h) \geq 0, \quad \forall i \in I, \forall h \in \mathcal{H}_i \tag{126}$$

$$b_i(h) \in \{0, 1\}, \quad \forall h \in \mathcal{H}_i \tag{127}$$

$$y_i(\varphi) \in (-\infty, +\infty), \quad \forall i \in I, \forall \varphi \in \Phi_i \tag{128}$$

$$z(j) \in [0, 1], \quad \forall j \in \mathcal{E} \tag{129}$$

Table 5: **MILP-n agents**. This 0-1 mixed integer linear program, derived from game theoretic considerations, finds *pure* optimal joint policies for DEC-POMDPs with 3 or more agents.

Compared to the **MILP** of Table 3, **MILP-n agents** has roughly the same size but with more real valued variables and more 0-1 variables. To be precise, **MILP** has a 0-1 variable for every terminal history of every agent (that is *approximatively* $\sum_{i\in I} |A_i|^T |O_i|^{T-1}$ integer variables) while **MILP-n agents** has two 0-1 variables for every terminal as well as nonterminal history of each agent (*approximatively* $2\sum_{i\in I}(|A_i||O_i|)^T$ integer variables).

5.5 Summary

The formulation of the solution of a DEC-POMDP and the application of the Duality Theorem for Linear Programs allow us to formulate the solution of a DEC-POMDP as the solution of a new kind of 0-1 MILP. For 2 agents, this MILP has "only" $O(k^T)$ variables and constraints and is thus "smaller" than **MILP** of the previous section. Still, all these MILPS are quite large and the next section investigates heuristic ways to speed up their resolution.

6. Heuristics for Speeding up the Mathematical Programs

This section focusses on ways to speed up the resolution of the various MILPs presented so far. Two ideas are exploited. First, we show how to prune the set of sequence-form policies by removing histories that will provably not be part of the optimal joint policy. These histories are called "locally extraneous". Then, we give some lower and uppers bounds to the objective function of the MILPs, these bounds can sometimes be used in the "branch and bound" method often used by MILP solvers to finalize the values of the integer variables.

6.1 Locally Extraneous Histories

A locally extraneous history is a history that is not required to find an optimal joint policy when the initial state of the DEC-POMDP is α because it could be replaced by a *co-history* without affecting the value of the joint policy. A **co-history** of a history h of an agent is defined to be a history of that agent that is identical to h in all aspects *except* for its last action. If $A_i = \{b, c\}$, the only co-history of $c.u.b.v.b$ is the history $c.u.b.v.c$. The set of co-histories of a history h shall be denoted by $C(h)$.

Formally, a history $h \in \mathcal{H}_i^t$ of length t of agent i is said to be **locally extraneous** if, for every probability distribution γ over the set $\mathcal{H}_{-i}^t$ of i-reduced joint histories of length t, there exists a history $h' \in C(h)$ such that

$$\sum_{j'\in\mathcal{H}_{-i}^t} \gamma(j')\big\{\mathcal{R}(\alpha,\langle h', j'\rangle) - \mathcal{R}(\alpha,\langle h, j'\rangle)\big\} \quad \geq \quad 0 \tag{130}$$

where $\gamma(j')$ denotes the probability of j' in γ.

An alternative definition is as follows. A history $h \in \mathcal{H}_i^t$ of length t of agent i is said to be **locally extraneous** if there exists a probability distribution ω over the set of co-histories of h such that for each i-reduced joint history j' of length t, there holds

$$\sum_{h'\in C(h)} \omega(h')\mathcal{R}(\alpha,\langle h', j'\rangle) \geq \mathcal{R}(\alpha,\langle h, j'\rangle) \tag{131}$$

where $\omega(h')$ denotes the probability of the co-history h' in ω.

The following theorem justifies our incremental pruning of locally extraneous histories so that the search for optimal joint policies is faster because it is performed on a smaller set of possible support histories.

Theorem 6.1. *For every optimal T-period joint policy p' such that for some agent $i \in I$ and for a terminal history h of agent i that is locally extraneous at α, $p'_i(h) > 0$, there exists another T-period joint policy p that is optimal at α and that is identical to p' in all respects except that $p_i(h) = 0$.*

Proof: Let p' be a T-period joint policy that is optimal at α. Assume that for some agent $i \in I$ and for a terminal history h of agent i that is locally extraneous at α, $p'_i(h) > 0$. By (130), there exists at least one co-history h' of h such that,

$$\sum_{j' \in \mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha, \langle h', j'\rangle) - \mathcal{R}(\alpha, \langle h, j'\rangle)\big\} \quad \geq \quad 0. \tag{132}$$

Let q be a T-period policy of agent i that is identical to p'_i in all respects except that $q(h')$ $= p'_i(h) + p'_i(h')$ and $q(h) = 0$. We shall show that q is also optimal at α. There holds,

$$\begin{aligned}
\mathcal{V}(\alpha, \langle q, p'_{-i}\rangle) - \mathcal{V}(\alpha, \langle p'_i, p_{-i}\rangle) \quad &= \\
\sum_{j' \in \mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha, \langle h', j'\rangle)q(h') - \mathcal{R}(\alpha, \langle h', j'\rangle)p'_i(h') - \mathcal{R}(\alpha, \langle h, j'\rangle)p'_i(h)\big\} \quad &= \\
\sum_{j' \in \mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha, \langle h', j'\rangle)(q(h') - p'_i(h')) - \mathcal{R}(\alpha, \langle h, j'\rangle)p'_i(h)\big\} \quad &= \\
\sum_{j' \in \mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha, \langle h', j'\rangle)p'_i(h) - \mathcal{R}(\alpha, \langle h, j'\rangle)p'_i(h)\big\} &
\end{aligned}$$

since $q(h') = p'_i(h) + p'_i(h')$. Therefore,

$$\begin{aligned}
\mathcal{V}(\alpha, \langle q, p'_{-i}\rangle) - \mathcal{V}(\alpha, \langle p'_i, p_{-i}\rangle) \quad &= \\
\sum_{j' \in \mathcal{H}^T_{-i}} p'_{-i}(j')\big\{\mathcal{R}(\alpha, \langle h', j'\rangle) - \mathcal{R}(\alpha, \langle h, j'\rangle)\big\} \quad &\geq \quad 0 \quad \text{(due to (132))}.
\end{aligned}$$

Hence, $p = \langle q, p'_{-i}\rangle$ is also an optimal T-period joint policy at α. □

One could also wonder if the order with which extraneous histories are pruned is important or not. To answer this question, the following theorem shows that if many co-histories are extraneous, they can be pruned in any order as:

- either they all have the same value, so any one of them can be pruned ;
- or pruning one of them does not change the fact that the others are still extraneous.

Theorem 6.2. *If two co-histories h_1 and h_2 are both locally extraneous, either their values $\mathcal{R}(\alpha, \langle h_1, j'\rangle)$ and $\mathcal{R}(\alpha, \langle h_2, j'\rangle)$for all $j' \in \mathcal{H}^t_{-i}$ are equal or h_1 is also locally extraneous relatively to $C(h) \setminus \{h_2\}$.*

Proof: Let C^+ denotes the union $C(h_1) \cup C(h_2)$. We have immediately that $C(h_1) = C^+ \setminus \{h_1\}$ and $C(h_2) = C^+ \setminus \{h_2\}$. h_1 (resp. h_2) being locally extraneous means that there exists a probability distribution ω_1 on $C(h_1)$ (resp. ω_2 on $C(h_2)$) such that, for all j' of $\mathcal{H}^t_{-i}$:

$$\sum_{h' \in C^+ \setminus \{h_1\}} \omega_1(h') \mathcal{R}(\alpha, \langle h', j' \rangle) \geq \mathcal{R}(\alpha, \langle h_1, j' \rangle) \tag{133}$$

$$\sum_{h' \in C^+ \setminus \{h_2\}} \omega_2(h') \mathcal{R}(\alpha, \langle h', j' \rangle) \geq \mathcal{R}(\alpha, \langle h_2, j' \rangle) \tag{134}$$

$$\tag{135}$$

Eq. (133) can be expanded in:

$$\omega_1(h_2) \mathcal{R}(\alpha, \langle h_2, j' \rangle) + \sum_{h' \in C^+ \setminus \{h_1, h_2\}} \omega_1(h') \mathcal{R}(\alpha, \langle h', j' \rangle) \geq \mathcal{R}(\alpha, \langle h_1, j' \rangle). \tag{136}$$

Using (134) in (136) gives

$$\omega_1(h_2) \sum_{h' \in C^+ \setminus \{h_2\}} \omega_2(h') \mathcal{R}(\alpha, \langle h', j' \rangle) + \sum_{h' \in C^+ \setminus \{h_1, h_2\}} \omega_1(h') \mathcal{R}(\alpha, \langle h', j' \rangle) \geq \mathcal{R}(\alpha, \langle h_1, j' \rangle) \tag{137}$$

leading to

$$\sum_{h' \in C^+ \setminus \{h_1, h_2\}} (\omega_1(h_2)\omega_2(h') + \omega_1(h')) \mathcal{R}(\alpha, \langle h', j' \rangle) \geq (1 - \omega_1(h_2)\omega_2(h_1)) \mathcal{R}(\alpha, \langle h_1, j' \rangle) \tag{138}$$

So, two cases are possible:

- $\omega_1(h_2) = \omega_2(h_1) = 1$. In that case, as $\mathcal{R}(\alpha, \langle h_2, j' \rangle) \geq \mathcal{R}(\alpha, \langle h_1, j' \rangle)$ and $\mathcal{R}(\alpha, \langle h_1, j' \rangle) \geq \mathcal{R}(\alpha, \langle h_2, j' \rangle)$, we have that $\mathcal{R}(\alpha, \langle h_1, j' \rangle) = \mathcal{R}(\alpha, \langle h_2, j' \rangle)$ for all j' of $\mathcal{H}^t_{-i}$.
- $\omega_1(h_2)\omega_2(h_1) < 1$. In that case we have:

$$\sum_{h' \in C^+ \setminus \{h_1, h_2\}} \frac{\omega_1(h_2)\omega_2(h') + \omega_1(h')}{1 - \omega_1(h_2)\omega_2(h_1)} \mathcal{R}(\alpha, \langle h', j' \rangle) \geq \mathcal{R}(\alpha, \langle h_1, j' \rangle) \tag{139}$$

meaning that even without using h_2, h_1 is still locally extraneous because $\frac{\omega_1(h_2)\omega_2(h') + \omega_1(h')}{1 - \omega_1(h_2)\omega_2(h_1)}$ is a probability distribution over $C^+ \setminus \{h_1, h_2\}$

$$\sum_{h' \in C^+ \setminus \{h_1, h_2\}} \frac{\omega_1(h_2)\omega_2(h') + \omega_1(h')}{1 - \omega_1(h_2)\omega_2(h_1)} = \frac{\omega_1(h_2)(1 - \omega_2(h_1)) + (1 - \omega_1(h_2))}{1 - \omega_1(h_2)\omega_2(h_1)} \tag{140}$$

$$= \frac{1 - \omega_1(h_2)\omega_2(h_1)}{1 - \omega_1(h_2)\omega_2(h_1)} \tag{141}$$

$$= 1. \tag{142}$$

□

In order to prune locally extraneous histories, one must be able to identify these histories. There are indeed two complementary ways for doing this.

The first method relies on the definition of the value of a history (see Section 3.3), that is

$$\mathcal{R}(\alpha, \langle h, j' \rangle) \quad = \quad \Psi(\alpha, \langle h, j' \rangle) \overline{R}(\alpha, \langle h, j' \rangle). \tag{143}$$

Therefore, if

$$\Psi(\alpha, \langle h, j' \rangle) \quad = \quad 0, \quad \forall j' \in \mathcal{H}^t_{-i} \tag{144}$$

is true for a history h, then that means that every joint history of length t occurring from α of which the given history is a part of has an *a priori* probability of 0. thus, h is clearly extraneous. Besides, every co-history of h will also be locally extraneous as they share the same probabilities.

A second test is needed because some locally extraneous histories do not verify (144). Once again, we turn to linear programing and in particular to the following linear program

Variables: $y(j)$, $\forall j \in \mathcal{H}^t_{-i}$

$$\text{Minimize} \quad \epsilon \tag{145}$$

subject to:

$$\sum_{j' \in \mathcal{H}^t_{-i}} y(j') \big\{ \mathcal{R}(\alpha, \langle h', j' \rangle) - \mathcal{R}(\alpha, \langle h, j' \rangle) \big\} \quad \leq \quad \epsilon, \quad \forall h' \in C(h) \tag{146}$$

$$\sum_{j' \in \mathcal{H}^t_{-i}} y(j') \quad = \quad 1 \tag{147}$$

$$y(j') \quad \geq \quad 0, \quad \forall j' \in \mathcal{H}^t_{-i} \tag{148}$$

because of the following Lemma.

Lemma 6.1. *If, it exists a solution (ϵ^*, y^*) to the linear program* (145)-(148) *where $\epsilon^* \geq 0$, then h is locally extraneous.*

Proof: Let (ϵ^*, y^*) be a solution to the LP (145)-(148). y^* is a probability distribution over $\mathcal{H}^t_{-i}$ due to constraints (147)-(148). If $\epsilon^* \geq 0$, since we are minimizing ϵ, due to constraints (146), we have that for every $\tilde{y} \in \Delta(\mathcal{H}^t_{-i})$, and for every co-history h' of h

$$\sum_{j' \in \mathcal{H}^t_{-i}} \tilde{y}(j') \big\{ \mathcal{R}(\alpha, \langle h', j' \rangle) - \mathcal{R}(\alpha, \langle h, j' \rangle) \big\} \quad \geq \quad \epsilon^*. \tag{149}$$

Therefore, by definition, h is locally extraneous. □

The following procedure identifies all locally extraneous terminal histories of all the agents and proceed to their iterative pruning. This is mainly motivated by Theorems 6.1 and 6.2 for effectively removing extraneous histories. The procedure is similar to the procedure of iterated elimination of dominated strategies in a game (Osborne & Rubinstein, 1994). The concept is also quite similar to the process of policy elimination in the backward step of the dynamic programming for partially observable stochastic games (Hansen et al., 2004).

- **Step 1:** For each agent $i \in I$, set $\tilde{H}_i^T$ to $\mathcal{E}_i$. Let $\tilde{H}^T$ denote the set $\times_{i \in I} \tilde{H}_i^T$. For each joint history $j \in \tilde{H}^T$, compute and store the value $\mathcal{R}(\alpha, j)$ of j and the joint observation sequence probability $\Psi(\alpha, j)$ of j.
- **Step 2:** For each agent $i \in I$, for each history $h \in \tilde{H}_i^T$, if for each i-reduced joint history $j' \in \tilde{H}_{-i}^T$, $\Psi(\alpha, \langle h, j' \rangle) = 0$, remove h from $\tilde{H}_i^T$.
- **Step 3:** For each agent $i \in I$, for each history $h \in \tilde{H}_i^T$ do as follows: If $C(h) \cap \tilde{H}_i^T$ is non-empty, check whether h is locally extraneous or not by setting up and solving LP (145)-(148). When setting the LP, replace $\mathcal{H}_{-i}^t$ by the set $\tilde{H}_{-i}^T$ and the set $C(h)$ by the set $C(h) \cap \tilde{H}_i^T$. If upon solving the LP, h is found to be locally extraneous at α, remove h from $\tilde{H}_i^T$.
- **Step 4:** If in Step 3 a history (of any agent) is found to be locally extraneous, go to Step 3. Otherwise, terminate the procedure.

The procedure builds the set $\tilde{H}_i^T$ for each agent i. This set contains every terminal history of agent i that is required for finding an optimal joint policy at α, that is every terminal history that is not locally extraneous at α. For each agent i, every history that is in $\mathcal{H}_i^T$ but not in $\tilde{H}_i^T$ is locally extraneous. The reason for reiterating Step 3 is that if a history h of some agent i is found to be locally extraneous and consequently removed from $\tilde{H}_i^T$, it is possible that a history of some other agent that was previously not locally extraneous now becomes so, due to the removal of h from $\tilde{H}_i^T$. Hence, in order to verify if this is the case for any history or not, we reiterate Step 3.

Besides, Step 2 of the procedure below also prunes histories that are impossible given the model of the DEC-POMDP because their observation sequence can not be observed.

A last pruning step can be taken in order to remove non-terminal histories that can *only* lead to extraneous terminal histories. This last step is recursive, starting from histories of horizon $T-1$, we remove histories h_i that have no non-extraneous terminal histories, that is, histories h_i such that all $h.o.a$ are extraneous for $a \in A_i$ and $o \in O_i$.

Complexity The algorithm for pruning locally extraneous histories has an exponential complexity. Each joint history must be examined to compute its value and its occurence probability. Then, in the worst case, a Linear Program can be run for every local history in order to check it is extraneous or not. Experimentations are needed to see if the prunning is really interesting.

6.2 Cutting Planes

Previous heuristics were aimed at reducing the search space of the linear programs, which incidentally has a good impact on the time needed to solve these programs. Another option which directly aims at reducing the computation time is to use cutting planes (Cornuéjols, 2008). A cut (Dantzig, 1960) is a special constraint that identifies a portion of the set of feasible solutions in which the optimal solution provably does not lie. Cuts are used in conjunction with various "branch and bounds" mechanism to reduce the number of possibles combination of integer variables that are examined by a solver.

We will present two kinds of cuts.

Variables: $y(j)$, $\forall j \in \mathcal{H}$

$$\text{Maximize} \quad \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) y(j) \tag{153}$$

subject to,

$$\sum_{a \in A} y(a) = 1 \tag{154}$$

$$-y(j) + \sum_{a \in A} y(j.o.a) = 0, \quad \forall j \in \mathcal{N}, \forall o \in O \tag{155}$$

$$y(j) \geq 0, \quad \forall j \in \mathcal{H} \tag{156}$$

Table 6: **POMDP**. This linear program finds an optimal policy for a POMDP.

6.2.1 Upper Bound for the Objective Function

The first cut we propose is the **upper bound POMDP cut**. The value of an optimal T-period joint policy at α for a given DEC-POMDP is bounded from above by the value $\mathcal{V}_P^*$ of an optimal T-period policy at α for the POMDP derived from the DEC-POMDP. This derived POMDP is the DEC-POMDP but assuming a centralized controller (i.e. with only one agent using joint-actions).

A sequence-form representation of the POMDP is quite straightforward. Calling $\mathcal{H}$ the set $\cup_{t=1}^{T} \mathcal{H}^t$ of joint histories of lengths less than or equal to T and $\mathcal{N}$ the set $\mathcal{H} \backslash \mathcal{E}$ of non-terminal joint histories, a policy for POMDP with horizon T in sequence-form is a function q from $\mathcal{H}$ to $[0, 1]$ such that:

$$\sum_{a \in A} q(a) = 1 \tag{150}$$

$$-q(j) + \sum_{a \in A} q(j.o.a) = 0, \quad \forall j \in \mathcal{N}, \forall o \in O \tag{151}$$

The value $\mathcal{V}_P(\alpha, q)$ of a sequence-form policy q is then given by:

$$\mathcal{V}_P(\alpha, q) = \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) q(j) \tag{152}$$

Thereby, the solution y^* of the linear program of Table 6 is an optimal policy for the POMDP of horizon T and the optimal value of the POMDP is $\sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) y^*(j)$. So, the value $\mathcal{V}(\alpha, p^*)$ of the optimal joint policy $p^* = \langle p_1^*, p_2^*, \cdots, p_n^* \rangle$ of the DEC-POMDP is bounded by above by the value $\mathcal{V}_P(\alpha, q^*)$ of the associated POMDP.

Complexity The complexity of finding an upper bound is linked to the complexity of solving a POMDP which, as showed by Papadimitriou and Tsitsiklis (1987), can be PSPACE (*i.e.* require a memory that is polynomial in the size of the problem, leading to a possible exponential complexity in time). Once again, only experimentation can help us decide in which cases the upper bound cut is efficient.

6.2.2 Lower Bound for the Objective Function

In the case of DEC-POMDPs with non-negative reward, it is trivial to show that the value of a T-period optimal policy is bounded from below by the value of the $T-1$ horizon optimal value. So, in the general case, we have to take into account the lowest reward possible to compute this lower bound and we can say that:

$$\sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, j) z(j) \quad \geq \quad \mathcal{V}^{T-1}(\alpha) + \min_{a \in A} \min_{s \in S} R(s, a) \tag{157}$$

where $\mathcal{V}^{T-1}$ is the value of the optimal policy with horizon $T-1$. The reasoning leads to an iterated computation of DEC-POMDPs of longer and longer horizon, reminiscent of the MAA* algorithm (Szer et al., 2005). Experiments will tell if it is worthwhile to solve bigger and bigger DEC-POMDPs to take advantage of a lower bound or if it is better to directly tackle the T horizon problem without using any lower bound.

Complexity To compute the lower bound, one is required to solve a DEC-POMDP whith an horizon that is one step shorter than the current horizon. The complexity is clearly at least exponential. In our experiments, the value of a DEC-POMDP has been used for the same DEC-POMDP with a bigger horizon. In such case, the computation time has been augmented by the best time to solve the smaller DEC-POMDP.

6.3 Summary

Pruning locally extraneous histories and using the bounds of the objective function can be of practical use for software solving the MILPs presented in this paper. Pruning histories means that the space of policies used by the MILP is reduced and, because the formulation of the MILP depends on combinatorial characteristics of the DEC-POMDP, these MILP must be altered as show in Appendix D.

Validity As far as cuts are concerned, they do not alter the solution found by the MILPs, so a solution to these MILPs is still an optimal solution to the DEC-POMDP. When extraneous histories are pruned, at least one valid policy is left as a solution because, in step 3 of the algorithm, an history is pruned only if it has other co-histories left. Besides, this reduced set of histories can still be used to build an optimal policy because of Theroem 6.1. As a consequence, the MILP build on this reduced set of histories admit a solution and this solution is one optimal joint policy.

In the next section, experimental results will allow us to understand in which cases the heuristics introduced can be useful.

7. Experiments

The mathematical programs and the heuristics designed in this paper are tested on four classical problems found in the literature. For these problems, involving *two* agents, we have mainly compared the computation time required to solve a DEC-POMDP using Mixed Integer Linear Programming methods to computation time reported for methods found in the literature. Then we have tested our programs on *three*-agent problems randomly designed.

Problem	$\|A_i\|$	$\|O_i\|$	$\|S\|$	n
MABC	2	2	4	2
MA-Tiger	3	2	2	2
Fire Fighting	3	2	27	2
Grid Meeting	5	2	16	2
Random Pbs	2	2	50	3

Table 7: "Complexity" of the various problems used as test beds.

MILP and **MILP-2** are solved using the "iLog Cplex 10" solver – a commercial set of Java packages – that relies on a combination of the "Simplex" and "Branch and Bounds" methods (Fletcher, 1987). The software is run on an Intel P4 at 3.4 GHz with 2Gb of RAM using default configuration parameters. For the mathematical programs, different combination of heuristics have been evaluated: pruning of locally extraneous histories, using a lower bound cut and using an upper bound cut, respectively denoted "LOC", "Low" and "Up" in the result tables to come.

The Non-Linear Program (**NLP**) of Section 3.4 has been evaluated by using various solvers from the NEOS website (`http://www-neos.mcs.anl.gov`), even thought this method does not guarantee an optimal solution to the DEC-POMDP. Three solvers have been used: LANCELOT (abbreviated as LANC.), LOQO and SNOPT.

The result tables also report results found in the literature for the following algorithms: DP stands for Dynamic Programming from Hansen et al. (2004); DP-LPC is an improved version of Dynamic Programming where policies are compressed in order to fit more of them in memory and speed up their evaluation as proposed by Boularias and Chaib-draa (2008); PBDP is an extension of Dynamic Programming where pruning is guided by the knowledge of reachable belief-states as detailed in the work of Szer and Charpillet (2006); MAA* is a heuristically guided forward search proposed by Szer et al. (2005) and a generalized and improved version of this algorithm called GMAA* developed by Oliehoek et al. (2008).

The problems selected to evaluate the algorithms are detailed in the coming subsections. They have been widely used to evaluate DEC-POMDPs algorithms in the literature and their "complexity", in term of space size, is summarized in Table 7.

7.1 Multi-Access Broadcast Channel Problem

Several versions of the Multi-Access Broadcast Channel (MABC) problem can be found in the literature. We will use the description given by Hansen et al. (2004) that allows this problem to be formalized as a DEC-POMDP.

In the MABC, we are given two nodes (computers) which are required to send messages to each other over a common channel for a given duration of time. Time is imagined to be split into discrete periods. Each node has a buffer with a capacity of one message. A buffer that is empty in a period is refilled with a certain probability in the next period. In a period, only one node can send a message. If both nodes send a message in the same period, a collision of the messages occurs and neither message is transmitted. In case of a collision, each node is intimated about it through a collision signal. But the collision

signaling mechanism is faulty. In case of a collision, with a certain probability, it does not send a signal to either one or both nodes.

We are interested in pre-allocating the channel amongst the two nodes for a given number of periods. The pre-allocation consists of giving the channel to one or both nodes in a period as a function of the node's information in that period. A node's information in a period consists only of the sequence of collision signals it has received till that period.

In modeling this problem as a DEC-POMDP, we obtain a 2-agent, 4-state, 2-actions-per-agent, 2-observations-per-agent DEC-POMDP whose components are as follows.

- Each node is an agent.
- The state of the problem is described by the states of the buffers of the two nodes. The state of a buffer is either Empty or Full. Hence, the problem has four states: (Empty, Empty), (Empty, Full), (Full, Empty) and (Full, Full).
- Each node has two possible actions, Use Channel and Don't Use Channel.
- In a period, a node may either receive a collision signal or it may not. So each node has two possible observations, Collision and No Collision.

The initial state of the problem α is (Full, Full). The state transition function $\mathbb{P}$, the joint observation function $\mathbb{G}$ and the reward function R have been taken from Hansen et al. (2004). If both agents have full buffers in a period, and both use the channel in that period, the state of the problem is unchanged in the next period; both agents have full buffers in the next period. If an agent has a full buffer in a period and only he uses the channel in that period, then his buffer is refilled with a certain probability in the next period. For agent 1, this probability is 0.9 and for agent 2, this probability is 0.1. If both agents have empty buffers in a period, irrespective of the actions they take in that period, their buffers get refilled with probabilities 0.9 (for agent 1) and 0.1 (for agent 2).

The observation function $\mathbb{G}$ is as follows. If the state in a period is (Full, Full) and the joint action taken by the agents in the previous period is (Use Channel, Use Channel), the probability that both receive a collision signal is 0.81, the probability that only one of them receives a collision signal is 0.09 and the probability that neither of them receives a collision signal is 0.01. For any other state the problem may be in a period and for any other joint action the agents may have taken in the previous period, the agents do not receive a collision signal.

The reward function R is quite simple. If the state in a period is (Full, Empty) and the joint action taken is (Use Channel, Don't Use Channel) or if the state in a period is (Empty, Full) and the joint action taken is (Don't Use Channel, Use Channel), the reward is 1; for any other combination of state and joint action, the reward is 0.

We have evaluated the various algorithms on this problem for three different horizons (3, 4 and 5) and the respective optimal policies have a value of 2.99, 3.89 and 4.79. Results are detailed in Table 8 where, for each horizon and algorithm, the value and the computation time for the best policy found are given.

The results show that the MILP compares favorably to more classical algorithms except for GMAA* that is always far better for horizon 4 and, for horizon 5, roughly within the

Resolution method			Horizon 3		Horizon 4		Horizon 5	
Program	Solver	Heuristics	Value	Time	Value	Time	Value	Time
MILP	Cplex	-	2.99	0.86	3.89	900	-	-m
MILP	Cplex	Low	2.99	0.10 / 0.93	3.89	0.39 / 900	-	3.5 / -m
MILP	Cplex	Up	2.99	0.28 / 1.03	3.89	0.56 / 907	-	4.73 / -m
MILP	Cplex	LOC	2.99	0.34 / 0.84	3.89	1.05 / 80	-	2.27 / -t
MILP	Cplex	LOC, Low	2.99	0.44 / 0.84	3.89	1.44 / 120	-	5.77 / -t
MILP	Cplex	LOC, Up	2.99	0.62 / 0.93	3.89	1.61 / 10.2	4.79	7.00 / 25
MILP-2	Cplex	-	2.99	0.39	3.89	3.53	-	-m
NLP	SNOPT	-	2.90	0.01	3.17	0.01	4.70	0.21
NLP	LANC.	-	2.99	0.02	3.79	0.95	4.69	20
NLP	LOQO	-	2.90	0.01	3.79	0.05	4.69	0.18
Algorithm	Family		Value	Time	Value	Time	Value	Time
DP	Dyn. Prog.		2.99	5	3.89	17.59	-	-m
DP-LPC	Dyn. Prog.		**2.99**	**0.36**	3.89	4.59	-	-m
PBDP	Dyn. Prog.		2.99	< 1s	3.89	2	4.79	10^5
MAA*	Fw. Search		2.99	< 1s	3.89	5400	-	-t
GMAA*	Fw. Search		?	?	**3.89**	**0.03**	**4.79**	**5.68**

Table 8: **MABC Problem**. Value and computation time (in seconds) for the solution of the problem as computed by several methods, best results are highlighted. When appropriate, time shows first the time used to run the heuristics then the global time, in the format `heuristic/total time`. "-t" means a timeout of 10,000s; "-m" indicates that the problem does not fit into memory and "?" indicates that the algorithm was not tested on that problem.

same order of magnitude as **MILP** with the more pertinent heuristics. As expected, apart for the simplest setting (horizon of 3), **NLP** based resolution can not find the optimal policy of the DEC-POMDP, but the computation time is lower than the other methods. Among MILP methods, **MILP-2** is better than **MILP** even with the best heuristics for horizon 3 and 4. When the size of the problem increases, heuristics are the only way for MILPs to be able to cope with the size of the problem. The table also shows that, for the MABC problem, pruning extraneous histories using the LOC heuristic is always a good method and further investigation revealed that 62% of the heuristics proved to be locally extraneous. As far are cutting bounds are concerned, they don't seem to be very useful at first (for horizon 3 and 4) but are necessary for **MILP** to find a solution for horizon 5. For this problem, one must also have in mind that there is only one optimal policy for each horizon.

7.2 Multi-Agent Tiger Problem

As explained in section 2.2, the Multi-Agent Tiger problem (MA-Tiger) has been introduced in the paper from Nair et al. (2003). From the general description of the problem, we ob-

Joint Action	State	Joint Observation	Probability
(Listen, Listen)	Left	(Noise Left, Noise Left)	0.7225
(Listen, Listen)	Left	(Noise Left, Noise Right)	0.1275
(Listen, Listen)	Left	(Noise Right, Noise Left)	0.1275
(Listen, Listen)	Left	(Noise Right, Noise Right)	0.0225
(Listen, Listen)	Right	(Noise Left, Noise Left)	0.0225
(Listen, Listen)	Right	(Noise Left, Noise Right)	0.1275
(Listen, Listen)	Right	(Noise Right, Noise Left)	0.1275
(Listen, Listen)	Right	(Noise Right, Noise Right)	0.7225
(*, *)	*	(*, *)	0.25

Table 9: Joint Observation Function $\mathbb{G}$ for the MA-Tiger Problem.

tain a 2-agent, 2-state, 3-actions-per-agent, 2-observations-per agent DEC-POMDP whose elements are as follows.

- Each person is an agent. So, we have a 2-agent DEC-POMDP.
- The state of the problem is described by the location of the tiger. Thus, S consists of two states Left (tiger is behind the left door) and Right (tiger is behind the right door).
- Each agent's set of actions consists of three actions: Open Left (open the left door), Open Right (open the right door) and Listen (listen).
- Each agent's set of observations consists of two observations: Noise Left (noise coming from the left door) and Noise Right (noise coming from the right door).

The initial state is an equi-probability distribution over S. The state transition function $\mathbb{P}$, joint observation function $\mathbb{G}$ and the reward function R are taken from the paper by Nair et al. (2003). $\mathbb{P}$ is quite simple. If one or both agents opens a door in a period, the state of the problem in the next period is set back to α. If both agents listen in a period, the state of the process in unchanged in the next period. $\mathbb{G}$, given in Table (9), is also quite simple. Nair et al. (2003) describes two reward functions called "A" and "B" for this problem, here we report only results for reward function "A", given in Table 10, as the behavior of the algorithm are similar for both reward functions. The optimal value of this problem for horizons 3 and 4 are respectively 5.19 and 4.80.

For horizon 3, dynamic programming or forward search methods are generally better than mathematical programs. But this is the contrary for horizon 4 were the computation time of **MILP** with the "Low" heuristic is significatively better than any other, even GMAA*. Unlike MABC, the pruning of extraneous histories does not improve methods based on MILP, this is quite understandable as deeper investigations showed that there are *no* extraneous histories. Using lower cutting bounds proves to be very efficient and can be seen as a kind of heuristic search for the best policy ; not directly in the set of policies (like

Joint Action	Left	Right
(Open Right, Open Right)	20	-50
(Open Left, Open Left)	-50	20
(Open Right, Open Left)	-100	-100
(Open Left, Open Right)	-100	-100
(Listen, Listen)	-2	-2
(Listen, Open Right)	9	-101
(Open Right, Listen)	9	-101
(Listen, Open Left)	-101	9
(Open Left, Listen)	-101	9

Table 10: Reward Function "A" for the MA-Tiger Problem.

GMAA*) but in the set of combination of histories, which may explain the good behavior of **MILP**+Low.

It must also be noted that for this problem, approximate methods like **NLP** but also other algorithms not depicted here like the "Memory Bound Dynamic Programming" of Seuken and Zilberstein (2007) are able to find the optimal solution. And, once again, methods based on a NLP are quite fast and sometimes very accurate.

7.3 Fire Fighters Problem

The problem of the Fire Fighters (FF) has been introduced as a new benchmark by Oliehoek et al. (2008). It models a team of n fire fighters that have to extinguish fires in a row of n_h houses.

The state of each house is given by an integer parameter, called the fire level f, that takes discrete value between 0 (no fire) and n_f (fire of maximum severity). At every time step, each agent can move to any one house. If two agents are at the same house, they extinguish any existing fire in that house. If an agent is alone, the fire level is lowered with a 0.6 probability if a neighbor house is also burning or with a 1 probability otherwise. A burning house with no fireman present will increase its fire level f by one point with a 0.8 probability if a neighbor house is also burning or with a probability of 0.4 otherwise. An unattended non-burning house can catch fire with a probability of 0.8 if a neighbor house is burning. After an action, the agents receive a reward of $-f$ for each house that is still burning. Each agent can only observe if there are flames at its location with a probability that depends on the fire level: 0.2 if $f = 0$, 0.5 if $f = 1$ and 0.8 otherwise. At start, the agents are outside any of the houses and the fire level of the houses is sampled from a uniform distribution.

The model has the following characteristics:

- n_a agents, each with n_h actions and n_f possible informations.
- There are $n_f^{n_h}.\binom{n_a+n_h-1}{n_a}$ states as there are $n_f^{n_h}$ possible states for the burning houses and $\binom{n_a+n_h-1}{n_a}$ different ways to distribute the n_a fire fighters in the houses. For example, 2 agents with 3 houses and 3 levels of fire lead to $9 \times 6 = 54$ states. But, it

Resolution method			Horizon 3		Horizon 4	
Program	Solver	Heuristics	Value	Time	Value	Time
MILP	Cplex	-	5.19	3.17	-	-t
MILP	Cplex	Low	5.19	0.46 / 4.9	**4.80**	**3.5 / 72**
MILP	Cplex	Up	5.19	0.42 / 3.5	-	0.75 / -t
MILP	Cplex	LOC	5.19	1.41 / 6.4	-	16.0 / -t
MILP	Cplex	LOC, Low	5.19	1.88 / 7.6	4.80	19.5 / 175
MILP	Cplex	LOC, Up	5.19	1.83 / 6.2	-	16.75 / -t
MILP-2	Cplex	-	5.19	11.16	-	-t
NLP	SNOPT	-	-45	0.03	-9.80	4.62
NLP	LANC.	-	5.19	0.47	4.80	514
NLP	LOQO	-	5.19	0.01	4.78	91
Algorithm	Family		Value	Time	Value	Time
DP	Dyn. Prog.		5.19	2.29	-	-m
DP-LPC	Dyn. Prog.		5.19	1.79	4.80	534
PBDP	Dyn. Prog.		?	?	?	?
MAA*	Fw. Search		**5.19**	**0.02**	4.80	5961
GMAA*	Fw. Search		5.19	0.04	4.80	3208

Table 11: **MA-Tiger Problem**. Value and computation time (in seconds) for the solution of the problem as computed by several methods, best results are highlighted. When appropriate, time shows first the time used to run the heuristics then the global time, in the format `heuristic/total time`."-t" means a timeout of 10.000s; "-m" indicates that the problem does not fit into memory and "?" indicates that the algorithm was not tested on that problem.

Resolution method			Horizon 3		Horizon 4	
Program	Solver	Heuristics	Value	Time	Value	Time
MILP	Cplex	-	-	-t	-	-t
MILP-2	Cplex	-	-5.98	38	-	-t
NLP	SNOPT	-	-5.98	0.05	-7.08	4.61
NLP	LANC.	-	-5.98	2.49	-7.13	1637
NLP	LOQO	-	-6.08	0.24	-7.14	83
Algorithm	Family		Value	Time	Value	Time
MAA*	Fw. Search		**(-5.73)**	**0.29**	(-6.57)	5737
GMAA*	Fw. Search		(-5.73)	0.41	**(-6.57)**	**5510**

Table 12: **Fire Fighting Problem**. Value and computation time (in seconds) for the solution of the problem as computed by several methods, best results are highlighted. "-t" means a timeout of 10.000s. For MAA* and GMAA*, value in parenthesis are taken from the work of Oliehoek et al. (2008) and *should* be optimal but are different from *our* optimal values.

is possible to use the information from the joint action to reduce the number of state needed in the transition function to simply $n_f^{n_h}$, meaning only 27 states for 2 agents with 3 houses and 3 levels of fire.

- Transition, observation and reward functions are easily derived from the above description.

For this problem, dynamic programming based methods are not tested as the problem formulation is quite new. For horizon 3, the value of the optimal policy given by Oliehoek et al. (2008) (-5.73) differs from the value found by the MILP algorithms (-5.98) whereas both methods are supposed to be exact. This might come from slight differences in our respective formulation of the problems. For horizon 4, Oliehoek et al. (2008) report an optimal value of (-6.57).

For this problem, MILP methods are clearly outperformed by MAA* and GMAA*. Only NLP methods, which give an optimal solution for horizon 3, are better in term of computation time. It might be that NLP are also able to find optimal policies for horizon 4 but as our setting differs from the work of Oliehoek et al. (2008), we are not able to check if the policy found is really the optimal. The main reason for the superiority of forward search method lies in the fact that this problem admits many many optimal policies with the same value. In fact, for horizon 4, MILP-based methods find an optimal policy quite quickly (around 82s for **MILP-2**) but then, using branch-and-bound, must evaluate all the other potential policies before knowing that it indeed found an optimal policy. Forward search methods stop nearly as soon as they hit one optimal solution.

Heuristics are not reported as, not only do they not improve the performance of MILP but they take away some computation time and thus the results are worse.

7.4 Meeting on a Grid

The problem called "Meeting on a grid" deals with two agents that want to meet and stay together in a grid world. It has been introduced in the work of Bernstein, Hansen, and Zilberstein (2005).

In this problem, we have two robots navigating on a two-by-two grid world with no obstacles. Each robot can only sense whether there are walls to its left or right, and the goal is for the robots to spend as much time as possible on the same square. The actions are to move up, down, left or right, or to stay on the same square. When a robot attempts to move to an open square, it only goes in the intended direction with probability 0.6, otherwise it randomly either goes in another direction or stays in the same square. Any move into a wall results in staying in the same square. The robots do not interfere with each other and cannot sense each other. The reward is 1 when the agents share a square, and 0 otherwise. The initial state distribution is deterministic, placing both robots in the upper left corner of the grid.

The problem is modelled as a DEC-POMDP where:

- There are 2 agents, each one with 5 actions and observations (wall on left, wall on right).
- There are 16 states, since each robot can be in any of 4 squares at any time.
- Transition, observation and reward functions are easily derived from the above description.

For this problem, dynamic programming based methods are not tested as the problem formulation is quite new. This problem is intrinsically more complex that FF and as such is only solved for horizon 2 and 3. Again, optimal value found by our method differ from the value reported by Oliehoek et al. (2008). Whereas we found that the optimal values are 1.12 and 1.87 for horizon 2 and 3, they report optimal values of 0.91 and 1.55.

Results for this problem have roughly the same pattern that the results for the FF problem. MAA* and GMAA* are quicker than MILP, but this time **MILP** is able to find an optimal solution for horizon 3. NLP methods give quite good results but they are slower than GMAA*. As for the FF, there are numerous optimal policies and MILP methods are not able to detect that the policy found quickly is indeed optimal.

Again, heuristics are not reported as, not only do they not improve the performance of MILP but they take away some computation time and thus the results are worse.

7.5 Random 3-Agent Problems

To test our approach on problems with 3 agents, we have used randomly generated DEC-POMDPs where the state transition function, the joint observation function and the reward functions are randomly generated. The DEC-POMDPs have 2 actions and 2 observations per agent and 50 states. Rewards are randomly generated integers in the range 1 to 5. The complexity of this family of problem is quite similar to the complexity of the MABC problem (see Section 7.1).

Resolution method			Horizon 2		Horizon 3	
Program	Solver	Heuristics	Value	Time	Value	Time
MILP	Cplex	-	1.12	0.65	1.87	1624
MILP-2	Cplex	-	1.12	0.61	-	-t
NLP	SNOPT	-	0.91	0.01	1.26	1.05
NLP	LANC.	-	1.12	0.06	1.87	257
NLP	LOQO	-	1.12	0.07	0.48	81
Algorithm	Family		Value	Time	Value	Time
MAA*	Fw. Search		**(0.91)**	**0s**	(1.55)	10.8
GMAA*	Fw. Search		**(0.91)**	**0s**	**(1.55)**	**5.81**

Table 13: **Meeting on a Grid Problem**. Value and computation time (in seconds) for the solution of the problem as computed by several methods, best results are highlighted. "-t" means a timeout of 10.000s. For MAA* and GMAA*, value in parenthesis are taken from the work of Oliehoek et al. (2008) and *should* be optimal but are different from *our* optimal values...

Program	Least Time (secs)	Most Time (secs)	Average	Std. Deviation
MILP	2.45	455	120.6	183.48
MILP-2	6.85	356	86.88	111.56

Table 14: Times taken by **MILP** and **MILP-2** on the 2-agent Random Problem for horizon 4.

In order to assess the "real" complexity of this Random problem, we have first tested a two-agent version of the problem for a horizon of 4. Results averaged over 10 runs of the programs are given in Table 14. When compared to the MABC problem which seemed of comparable complexity, the Random problem proves easier to solve (120s vs 900s). For this problem, the number of 0-1 variable is relatively small, as such it does not weight too much on the resolution time of **MILP-2** which is thus faster.

Results for a three-agent problem with horizon 3 are given in Table 15, once again averaged over 10 runs. Even though the size of the search space is "smaller" in that case (for 3 agents and a horizon of 3, there are 9×10^{21} policies whereas the problem with 2 agents and horizon 4, there are 1.5×10^{51} possible policies), the 3 agent problems seems more difficult to solve, demonstrating that one of the big issue is policy coordination. Here, heuristics bring a significative improvement on the resolution time of **MILP**. As predicted, **MILP-n** is not very efficient and is only given for completeness.

Program	Least Time (secs)	Most Time (secs)	Average	Std. Deviation
MILP	21	173	70.6	64.02
MILP-Low	26	90	53.2	24.2
MILP-n	754	2013	1173	715

Table 15: Times taken by **MILP** and **MILP-n** on the 3-agent Random problem for horizon 3.

8. Discussion

We have organized the discussion in two parts. In the first part, we analyze our results and offer explanations on the behavior of our algorithms and the usefulness of heuristics. Then, in a second part, we explicitely address some important questions.

8.1 Analysis of the Results

From the results, it appears that MILP methods are a better alternative to Dynamic Programming methods for solving DEC-POMDPs but are globally and generally clearly outperformed by forward search methods. The structure and thus the characteristics of the problem have a big influence on the efficiency of the MILP methods. Whereas it seems that the behavior of GMAA* in terms of computation time is quite correlated with the complexity of the problem (size of the action and observation spaces), MILP methods seem sometimes less correlated to this complexity. It is the case for the MABC problem (many extraneous histories can be pruned) and the MA-Tiger problem (special structure) where they outperform GMAA*. On the contrary, when many optimal policies exists, forward search methods like GMAA* are clearly a better choice. Finally, Non-Linear Programs, even though they can not guarantee an optimal solution, are generally a good alternative as they are sometimes able to find a very good solution and their computation time is often better than GMAA*. This might prove useful for approximate heuristic-driven forward searches.

The computational record of the two 2-agent programs shows that **MILP-2 agents** is slower than **MILP** when the horizon grows. There are two reasons to which the sluggishness of **MILP-2 agents** may be attributed. The time taken by the branch and bound (BB) method to solve a 0-1 MILP is inversely proportional to the number of 0-1 variables in the MILP. **MILP-2 agents** has many more 0-1 variables than **MILP** event hough the total number of variables in it is exponentially less than in **MILP**. This is the first reason. Secondly, **MILP-2 agents** is a more *complicated* program than **MILP**; it has many more constraints than **MILP**. **MILP** is a simple program, concerned only with finding a subset of a given set. In addition to finding weights of histories, **MILP** also finds weights of terminal joint histories. This is the only extra or superfluous quantity it is forced to find. On the other hand, **MILP-2 agents** takes a much more circuitous route, finding many more superfluous quantities than **MILP**. In addition to weights of histories, **MILP-2 agents** also finds supports of policies, regrets of histories and values of information sets. Thus, the

Problem	Heuristic	Horizon 2		Horizon 3		Horizon 4		Horizon 5	
		Time	#pruned	Time	#pruned	Time	#pruned	Time	#pruned
MABC	LOC			0.34	14/32	1.047	74/128	2.27	350/512
	Low			0.10		0.39		3.5	
	Up			0.28		3.89		4.73	
MA-Tiger	LOC	0.41	0/18	1.41	0/108	16.0	0/648		
	Low			0.46		3.5			
	Up			0.42		0.75			
Meeting	LOC	1.36	15/50*	74.721	191/500*				

Table 16: **Computation time of heuristics**. For the LOC heuristics, we give the computation time in seconds and the number of locally extraneous histories pruned over the total number of histories (for an agent). A '*' denotes cases where *one* additional history is prunned for the second agent. For the Low and Up heuristic, only computation time is given.

relaxation of **MILP-2 agents** takes longer to solve than the relaxation of **MILP**. This is the second reason for the slowness with which the BB method solves **MILP-2 agents**.

For bigger problems, namely Fire-Fighters and Meeting on a Grid, when the horizon stays small, **MILP-2 agents** can compete with **MILP** because of its slightly lower size. Its complexity grows like $O((|A_i||O_i|)^T)$ whereas it grows like $O((|A_i||O_i|)^{2T})$ for **MILP**. But that small difference does not hold long as the number of integer variables quickly lessens the efficiency of **MILP-2 agents**.

As far as heuristic are concerned, they proved to be invaluable for some problems (MABC and MA-Tiger) and useless for others. In the case of MABC, heuristics are very helpful to prune a large number of extraneous heuristics but ultimately, it is the combination with the upper bound cut that it the more efficient when the horizon grows. In the case of MA-Tiger, although no extraneous histories are found, using the lower bound cut heuristic with **MILP** leads to the quickest algorithm for solving the problem with a horizon of 4. For other problems, heuristics are more of a burden as they are too greedy in computation time to speed up the resolution. For example, for the "Grid Meeting" problem, the time taken to prune extraneous histories is bigger than the time saved for solving the problem.

As a result, the added value of using heuristics depends on the nature of the problem (as depicted in Table 16) but, right now, we are not able to predict their usefulness without trying them.

We also emphasize that the results given here lie at the limit of what is possible to solve in an exact manner given the memory of the computer used for the resolution, especially in terms of the horizon. Furthermore, as the number of agent increases, the length of the horizon must be decreased for the problems to still be solvable.

8.2 Questions

The mathematical programing approach presented in this paper raises different questions. We have explicitly addressed some of the questions that appears important to us.

Q1: Why is the sequence-form approach not entirely doomed by its exponential complexity?

As the number of sequence-form *joint* policies grows doubly exponentially with the horizon and the number of agents, the sequence-form approach seems doomed, even compared to dynamic programming which is doubly exponential in the worst cases only. But, indeed, some arguments must be taken into consideration.

"Only" an exponential number of *individual* histories need to be evaluated. The "joint" part of the sequence-form is left to the MILP solver. And every computation done on a particular history, like computing its value or checking if it is extraneous, has a greater "reusability" than computations done on entire policies. An history is shared by many more joint policies than an individual policy. In some way, sequence-form allows us to work on reusable part of policies without having to work directly in the world of distributions on the set of joint-policies.

Then, the MILPs derived from the sequence-form DEC-POMDPs need a memory size which grows "only" exponentially with the horizon and the number of agents. Obviously, such a complexity is quickly overwhelming but it is also the case of every other exact method so far. As shown by the experiments, the MILP approach derived from the sequence-form compares quite well with dynamic programming, even if outperformed by forward methods like GMAA*.

Q2: Why does MILP sometimes take so little time to find an optimal joint policy when compared to existing algorithms?

Despite the complexity of our MILP approach, three factors contribute to the relative efficiency of **MILP**.

1. First, the efficiency of linear programming tools themselves. In solving **MILP**, the BB method solves a sequence of linear programs using the simplex algorithm. Each of these LPs is a relaxation of **MILP**. In theory, the simplex algorithm requires in the worst case an exponential number of steps (in the size of the LP) in solving a LP[2], but it is well known that, in practice, it usually solves a LP in a polynomial number of steps (in the size of the LP). Since the size of a relaxation of **MILP** is exponential in the horizon, this means that, roughly speaking, the time taken to solve a relaxation of **MILP** is "only" exponential in the horizon whereas it can be doubly exponential for other methods.

2. The second factor is the sparsity of the matrix of coefficients of the constraints of **MILP**. The sparsity of the matrix formed by the coefficients of the constraints of

2. This statement must be qualified: this worst case time requirement has not been demonstrated for all *variants* of the simplex algorithm. It has been demonstrated only for the basic version of the simplex algorithm.

an LP determines in practice the rate with which a pivoting algorithm such as the simplex solves the LP (this also applies to Lemke's algorithm in the context of an LCP). The sparser this matrix, the lesser the time required to perform elementary pivoting (row) operations involved in the simplex algorithm and the lesser the space required to model the LP.

3. The third factor is the fact that we supplement **MILP** with cuts; the computational experience clearly shows how this speeds up the computations. While the first two factors were related to solving a relaxation of **MILP** (i.e., an LP), this third factor has an impact on the BB method itself. The upper bound cut identifies an additional terminating condition for the BB method, thus enabling it to terminate earlier than in the absence of this condition. The lower bound cut attempts to shorten the list of active subproblems (LPs) which the BB method solves sequentially. Due to this cut, the BB method has potentially a lesser number of LPs to solve. Note that in inserting the lower bound cut, we are emulating the forward search properties of the A* algorithm.

Q3: How do we know that the MILP-solver (iLog's "Cplex" in our experiments) is not the only reason for the speedup?

Clearly, our approach would be slower, even sometime slower than a classical dynamic programming approach if we had used another program for solving our MILPs as we experimented also our MILPs with solvers from the NEOS website that were indeed very very slow. It is true that Cplex, the solver we have used in our experiments, is quite optimized. Nevertheless, it is exactly one of the points we wanted to experiment with in this paper: one of the advantages of formulating a DEC-POMDP as a MILP is the possibility to use the fact that, as mixed integer linear programs are very important for the industrial world, optimized solvers *do* exist.
Then, we *had* to formulate a DEC-POMDP as a MILP and this is mostly what this paper is about.

Q4: What is the main contribution of this paper?

As stated earlier in the paper, current algorithms for DEC-POMDPs were largely inspired by POMDPs algorithms. Our main contribution was to pursue an entirely different approach, *i.e.*, mixed integer linear programming. As such, we have learned a lot about DEC-POMDPs and about the *pro & con* of this mathematical programming approach. This has lead to the formulation of new algorithms.

In designing these algorithms, we have, first of all, drawn attention to a new representation of a policy, namely the sequence form of a policy, introduced by Koller, Megiddo and von Stengel. The **sequence form of a policy** is *not* a compact representation of the policy of an agent, but it does afford a compact representation of the *set* of policies of the agent.

The algorithms we have proposed for finite horizon DEC-POMDPs are **mathematical programming algorithms**. To be precise, they are 0-1 MILPs. In the MDP domain,

mathematical programming has been long used for solving the infinite horizon case. For instance, an infinite horizon MDP can be solved by a linear program (d'Epenoux, 1963). More recently, mathematical programming has been directed at infinite horizon POMDPs and DEC-POMDPs. Thus, an infinite horizon DEC-MDP (with state transition independence) can be solved by a 0-1 MILP (Petrik & Zilberstein, 2007) and an infinite horizon POMDP or DEC-POMDP can be solved (for local optima) by a nonlinear program (Amato, Bernstein, & Zilberstein, 2007b, 2007a). The finite horizon case – much different in character than the infinite horizon case – has been dealt with using dynamic programming. As stated earlier, whereas dynamic programming has been quite successful for finite horizon MDPs and POMDPs, it has been less so for finite horizon DEC-POMDPs.

In contrast, in game theory, mathematical programming has been successfully directed at games of finite horizon. Lemke's algorithm (1965) for two-player normal form games, the Govindan-Wilson algorithm (2001) for n-player normal form games and the Koller, Megiddo and von Stengel approach (which internally uses Lemke's algorithm) for two-player extensive form games are all for finite-horizon games.

What remained then was to a find way to appropriate mathematical programming for solving the finite horizon case of the POMDP/DEC-POMDP domain. Our work has done precisely this (incidently, we now have an algorithm for solving some kind of n-player normal form games). Throughout the paper, we have shown how mathematical programming (in particular, 0-1 integer programming) can be applied for solving finite horizon DEC-POMDPs (it is easy to see that the approach we have presented yields a linear program for solving a finite horizon POMDP). Additionally, the computational experience of our approach indicates that for finite horizon DEC-POMDPs, mathematical programming may be better (faster) than dynamic programming. We have also shown how the well-entrenched dynamic programming heuristic of the pruning of redundant or extraneous objects (in our case, histories) can be integrated into this mathematical programming approach.
Hence, the main contribution of this paper is that it presents, for the first time, an alternative approach for solving finite horizon POMDPs/DEC-POMDPs based on MILPs.

Q5: Is the mathematical programming approach presented in this paper something of a dead end?

This question is bit controversial and a very short answer to this question could be a "small yes". But this is true for every approach that looks for *exact* optimal solutions to DEC-POMDPs, whether it is grounded on dynamic programming or forward search or mathematical programming. Because of the complexity of the problem, an exact solution will always be untractable but our algorithms can still be improved.

A longer answer is more mitigated, especially in the light of the recent advances made for dynamic programming and forward search algorithms. One crucial point in sequence-form DEC-POMDPs is the pruning of extraneous histories. A recent work from Oliehoek, Whiteson, and Spaan (2009) has shown how to clusters histories that are equivalent in a way that could also reduce the nomber of constraints in MILPs. The approach of Amato, Dibangoye, and Zilberstein (2009) that improves and speed up the dynamic programming operator could help in finding extraneous histories. So, at the very least, some work is

still required before stating that every aspect of sequence-form DEC-POMDPs have been studied.

We now turn to an even longer answer. Consider the long horizon case. Given that exact algorithms (including the ones presented in this paper) can only tackle horizons less than 6, by 'long horizon', we mean anything upwards of 6 time periods. For the long horizon case, we are required to conceive a possibly sub-optimal joint policy for the given horizon and determine an upper bound on the loss of value incurred by using the joint policy instead of using an optimal joint policy.

The current trend for the long horizon case is a *memory-bounded* approach. The memory bounded dynamic programming (MBDP) algorithm (Seuken & Zilberstein, 2007) is the main exponent of this approach. This algorithm is based on the backward induction DP algorithm (Hansen et al., 2004). The algorithm attempts to run in a limited amount of space. In order to do so, unlike the DP algorithm, it prunes even non-extraneous (i.e., non-dominated) policy trees at each iteration. Thus, at each iteration, the algorithm retains a pre-determined number of trees. This algorithm and its variants have been used to find a joint policy for the MABC, the MA-tiger and the Box pushing problems for very long horizons (of the order of thousands of time periods).

MBDP does not provide an upper bound on the loss of value. The bounded DP (BDP) algorithm presented in the paper by Amato, Carlin, and Zilberstein (2007c) does give an upper bound. However, on more interesting DEC-POMDP problems (such as MA-tiger), MBDP finds a much better joint policy than BDP.

A meaningful way to introduce the notion of memory boundedness into our approach is to fix an *a priori* upper bound on the size of the concerned mathematical program. This presents all sorts of difficulties but the main difficulty seems to be the need to represent a policy for a long horizon in limited space. The MBDP algorithm solves this problem by using what may be termed as a recursive representation. The recursive representation causes the MBDP algorithm to take a long time to evaluate a joint policy, but it does allow the algorithm to represent a long horizon joint policy in limited space. In the context of our mathematical programming approach, we would have to change the policy constraints in some way so that a long horizon policy is represented by a system consisting of a limited number of linear equations and linear inequalities. Besides the policy constraints, other constraints of the presented programs would also have to be accordingly transfigured. It is not evident (to us) if such a transfiguration of the constraints is possible.

On the other hand, the infinite horizon case seems to be a promising candidate to adapt our approach to. Mathematical programming has already been applied, with some success, to solving infinite horizon DEC-POMDPs (Amato et al., 2007a). The computational experience of this mathematical programming approach shows that it is better (finds higher quality solutions in lesser time) than a dynamic programming approach (Bernstein et al., 2005; Szer & Charpillet, 2006).

Nevertheless, this approach has two inter-related shortcomings. First, the approach finds a joint controller (i.e., an infinite horizon joint policy) of a *fixed size* and not of the optimal size. Second, much graver than the first, for the fixed size, it finds a locally optimal joint controller. The approach does not guarantee finding an optimal joint controller. This is because the program presented in the work of Amato et al. (2007a) is a (non-convex)

nonlinear program (NLP). The NLP finds a fixed size joint controller in the canonical form (i.e., in the form of a finite state machine). We believe that both these shortcomings can be removed by conceiving a mathematical program (specifically, a 0-1 mixed integer linear program) that finds a joint controller in the sequence-form. As stated earlier, the main challenge in this regard is therefore an identification of the sequence-form of an infinite horizon policy. In fact, it may be that if such sequence-form characterization of an infinite horizon policy is obtained, it could be used in conceiving a program for the long horizon (undiscounted reward) case as well.

Q6: How does this help achieve designing artificial autonomous agents ?

At first sight, our work does not have any direct and immediate applied benefits for the purpose of building artificial intelligent agents or understanding how intelligence "works". Even in the limited field of multi-agent planning, our contributions are more on a theoretical level than on a practical one.

Real artificial multi-agent systems can indeed be modeled as DEC-POMDPs, even if they make use of communication, of common knowledge, of common social law. Then, such real systems would likely be made of a large number of states, actions or observations and require solutions over a large horizon. Our mathematical programming approach is practically useless in that setting as limited to DEC-POMDPs of very small size. Other models that are simpler – but far from trivial – to solve because they explicitly take into account some characteristics of the real systems do exist. Some works take advantage of communications (Xuan, Lesser, & Zilberstein, 2000; Ghavamzadeh & Mahadevan, 2004), some of the existing independencies in the system (Wu & Durfee, 2006; Becker, Zilberstein, Lesser, & Goldman, 2004), some do focus on interaction between agents (Thomas, Bourjot, & Chevrier, 2004), some, as said while answering the previous questions, rely on approximate solutions, *etc...* It is our intention to facilitate the re-use and the adaptation to these other models of the concepts used in our work and of the knowledge about the structure of an optimal solution of a DEC-POMDP. To that end, we decided not only to describe the MILP programs but also, and most importantly, *how* we derived these programs by making use of some properties of optimal DEC-POMDP solutions.

Truly autonomous agents will also require to adapt to new and unforeseen situations. Our work being dedicated to planning, it seems easy to argue that it does not contribute very much to that end either. On the other hand, learning in DEC-POMDPs has never really been addressed except for some fringe work in particular settings (Scherrer & Charpillet, 2002; Ghavamzadeh & Mahadevan, 2004; Buffet, Dutech, & Charpillet, 2007). In fact, even for "simple" POMDPs, learning is a very difficult task (Singh, Jaakkola, & Jordan, 1994). Currently, the more promising research deals with learning the "Predictive State Representation" (PSR) of a POMDP (Singh, Littman, Jong, Pardoe, & Stone, 2003; James & Singh, 2004; McCracken & Bowling, 2005). Making due allowance to the fundamental differences between the functional role of PSR and histories, we notice that PSR and histories are quite similar in structure. While it is too early to say, it might be that trying to learn the useful histories of a DEC-POMDP could take some inspiration from the way the right PSRs are learned for POMDPs.

9. Conclusion

We designed and investigated new exact algorithms for solving Decentralized Partially Observable Markov Decision Processes with finite horizon (DEC-POMDPs). The main contribution of our paper is the use of sequence-form policies, based on a sets of histories, in order to reformulate a DEC-POMDP as a non-linear programming problem (**NLP**). We have then presented two different approaches to linearize the **NLP** in order to find global and optimal solutions to DEC-POMDPs. The first approach is based on the combinatorial properties of the optimal policies of DEC-POMDPs and the second one relies on concepts borrowed from the field of game theory. Both lead to formulating DEC-POMDPs as *0-1 Mixed Integer Linear Programming problems* (MILPs). Several heuristics for speeding up the resolution of these MILPs make another important contribution of our work.

Experimental validation of the mathematical programming problems designed in this work was conducted on classical DEC-POMDP problems found in the literature. These experiments show that, as expected, our MILP methods outperform classical Dynamic Programming algorithms. But, in general, they are less efficient and more costly than forward search methods like GMAA*, especially in the case where the DEC-POMDP admits many optimal policies. Nevertheless, according to the nature of the problem, MILP methods can sometimes greatly outperform GMAA* (as in the MA-Tiger problem).

While it is clear that exact resolution of DEC-POMDPs can not scale up with the size of the problems or the length of the horizon, designing exact methods is useful in order to develop or improve approximate methods. We see at least three research directions where our work can contribute. One direction could be to take advantage of the large literature on algorithms for finding approximate solutions to MILPs and to adapt them to the MILPs formulated for DEC-POMDPs. Another direction would be to use the knowledge gained from our work to derive improved heuristics for guiding existing approximate existing methods for DEC-POMDPs. For example, the work of Seuken and Zilberstein (2007), in order to limit the memory resources used by the resolution algorithm, prune the space of policies to only consider some of them; our work could help using a better estimation of the policies that are important to be kept in the search space. Then, the one direction we are currently investigating is to adapt our approach to DEC-POMDPs of infinite length by looking for yet another representation that would allow such problems to be seen as MILPs.

More importantly, our work participates to a better understanding of DEC-POMDPs. We analyzed and understood key characteristics of the nature of optimal policies in order to design the MILPs presented in this paper. This knowledge can be useful for other work dealing with DEC-POMDPs and even POMDPs. The experimentations have also given some interesting insights on the nature of the various problems tested, in term of existence of extraneous histories or on the number of optimal policies. These insights might be a first step toward a taxonomy of DEC-POMDPs.

Appendix A. Non-Convex Non-Linear Program

Using the simplest example, this section aims at showing that the Non-Linear Program (**NLP**) expressed in Table 2 can be non-convex.

Let us consider an example with two agents, each one with 2 possible actions (a and b) that want to solve a horizon-1 decision problem. The set of possible joint-histories is then: $\langle a,a\rangle$, $\langle a,b\rangle$, $\langle b,a\rangle$ and $\langle b,b\rangle$. Then the NLP to solve is:

Variables: $x_1(a)$, $x_1(b)$, $x_2(a)$, $x_2(a)$

$$\text{Maximize} \quad \begin{aligned} &\mathcal{R}(\alpha,\langle a,a\rangle)x_1(a)x_2(a) + \mathcal{R}(\alpha,\langle a,b\rangle)x_1(a)x_2(b) \\ &+\mathcal{R}(\alpha,\langle b,a\rangle)x_1(b)x_2(a) + \mathcal{R}(\alpha,\langle b,b\rangle)x_1(b)x_2(b) \end{aligned} \tag{158}$$

subject to

$$\begin{aligned} x_1(a) + x_1(b) &= 1 \\ x_2(a) + x_2(b) &= 1 \\ x_1(a) \geq 0, &\quad x_1(b) \geq 0 \\ x_2(a) \geq 0, &\quad x_2(b) \geq 0 \end{aligned}$$

A matrix formulation of the objective function of eq. (158) would be $x^T.C.x$ with C and x of the following kind:

$$C = \begin{bmatrix} 0 & 0 & c & d \\ 0 & 0 & e & f \\ c & e & 0 & 0 \\ d & f & 0 & 0 \end{bmatrix} \qquad x = \begin{bmatrix} x_1(a) \\ x_1(b) \\ x_2(a) \\ x_2(b) \end{bmatrix}. \tag{159}$$

If λ is the eigen value of vector $v = [v_1\ v_2\ v_3\ v_4]^T$ then it is straightforward to show that $-\lambda$ is also an eigen value: $[-v_1\ -v_2\ v_3\ v_4]^T = -\lambda C.[v_1\ v_2\ -v_3\ -v_4]^T$. As a result, the matrix C, hessian of the objective function, is not positive-definite and thus the objective function is not convex.

Appendix B. Linear Program Duality

Every linear program (LP) has a converse linear program called its dual. The first LP is called the primal to distinguish it from its dual. If the primal maximizes a quantity, the dual minimizes the quantity. If there are n variables and m constraints in the primal, there are m variables and n constraints in the dual. Consider the following (primal) LP.

Variables: $x(i)$, $\forall i \in \{1, 2, \cdots, n\}$

$$\text{Maximize} \quad \sum_{i=1}^{n} c(i)x(i)$$

subject to:

$$\begin{aligned} \sum_{i=1}^{n} a(i,j)x(i) &= b(j), \quad j = 1, 2, \cdots, m \\ x(i) &\geq 0, \quad i = 1, 2, \cdots, n \end{aligned}$$

This primal LP has one variable $x(i)$ for each $i = 1$ to n. The data of the LP consists of numbers $c(i)$ for each $i = 1$ to n, the numbers $b(j)$ for each $j = 1$ to m and the numbers $a(i, j)$ for each $i = 1$ to n and for each $j = 1$ to m. The LP thus has n variables and m constraints. The dual of this LP is the following LP.

Variables: $y(j)$, $\forall j \in \{1, 2, \cdots, m'\}$

$$\text{Minimize} \quad \sum_{j=1}^{m'} b(j)y(j)$$

subject To:

$$\begin{aligned} \sum_{j=1}^{m'} a(i,j)y(j) &\geq c(i), \quad i = 1, 2, \cdots, n' \\ y(j) &\in (-\infty, +\infty), \quad j = 1, 2, \cdots, m' \end{aligned}$$

The dual LP has one variable $y(j)$ for each $j = 1$ to m. Each $y(j)$ variable is a **free** variable. That is, it is allowed to take any value in $\mathbb{R}$. The dual LP has m variables and n constraints.

The theorem of linear programming duality is as follows.

Theorem B.1. *(Luenberger, 1984) If either a primal LP or its dual LP has a finite optimal solution, then so does the other, and the corresponding values of the objective functions are equal.*

Applying this theorem to the primal-dual pair given above, there holds,

$$\sum_{i=1}^{n} c(i)x^*(i) = \sum_{j=1}^{m} b(j)y^*(j)$$

where x^* denotes an optimal solution to the primal and y^* denotes an optimal solution to the dual.

The theorem of complementary slackness is as follows.

Theorem B.2. *(Vanderbei, 2008) Suppose that x is feasible for a primal linear program and y is feasible for its dual. Let $(w_1, \cdots, w_m)$ denote the corresponding primal slack variables, and let $(z_1, \cdots, z_n)$ denote the corresponding dual slack variables. Then x and y are optimal for their respective problems if and only if*

$$\begin{aligned} x_j z_j &= 0 \qquad \text{for } j = 1, \cdots, n, \\ w_i y_i &= 0 \qquad \text{for } i = 1, \cdots, m. \end{aligned}$$

Appendix C. Regret for DEC-POMDPs

The **value of an information set** $\varphi \in \mathcal{I}_i$ of an agent i for a i-reduced joint policy q, denoted $\lambda_i^*(\varphi, q)$, is defined by:

$$\lambda_i^*(\varphi, q) \quad = \quad \max_{h \in \varphi} \sum_{j' \in \mathcal{E}_{-i}} \mathcal{R}(\alpha, \langle h, j' \rangle) q(j') \tag{160}$$

for any terminal information set and, if φ is non-terminal, by:

$$\lambda_i^*(\varphi, q) \quad = \quad \max_{h \in \varphi} \sum_{o \in O_i} \lambda_i^*(h.o, q) \tag{161}$$

Then, the **regret of a history** h for an agent i and for a i-reduced joint policy q, denoted $\mu_i(h, q)$, it is defined by:

$$\mu_i(h, q) \quad = \quad \lambda_i^*(\varphi(h), q) - \sum_{j' \in H_{-i}^T} \mathcal{R}(\alpha, \langle h, j' \rangle) q(j') \tag{162}$$

if h is terminal and, if h is non-terminal, by:

$$\mu_i(h, q) \quad = \quad \lambda_i^*(\varphi(h), q) - \sum_{o \in O_i} \lambda_i^*(h.o, q) \tag{163}$$

The concept of regret of the agent i, which is independant of the policy of the agent i, is very useful when looking for optimal policy because its optimal value is known: it is 0. It is thus easier to manipulate than the optimal value of a policy.

Appendix D. Program Changes Due to Optimizations

Pruning locally or globally extraneous histories reduces the size of the search space of the mathematical programs. Now, some constraints of the programs depend on the size of the search space, we must then alter some of these constraints.

Let denote by a "$\sim$" superscript the sets actually used in our program. For example, $\tilde{\mathcal{E}}_i$ will be the actual set of terminal histories of agent i, be it pruned of extraneous histories or not.

Programs **MILP** (Table 3) and **MILP-n agents** (Table 5) rely on the fact that the number of histories of a given length t in the support of a pure policy of each agent is fixed and equal to $|O_i|^{t-1}$. As it may not be the case with pruned sets, the following changes have to be made:

- The constraint (42) of **MILP** or (121) **MILP-n agents**, that is

$$\sum_{j \in \mathcal{E}} z(j) \quad = \quad \prod_{i \in I} |O_i|^{T-1}$$

must be replaced by

$$\sum_{j \in \tilde{\mathcal{E}}} z(j) \quad \leq \quad \prod_{i \in I} |O_i|^{T-1}. \tag{164}$$

- The set of constraints (41) of **MILP** or (120) of **MILP-n agents**, that is

$$\sum_{j' \in \mathcal{E}_{-i}} z(\langle h, j' \rangle) \quad = \quad \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall i \in I, \forall h \in \mathcal{E}_i$$

must be replaced by

$$\sum_{j' \in \tilde{\mathcal{E}}_{-i}} z(\langle h, j' \rangle) \quad \leq \quad \prod_{k \in I \setminus \{i\}} |O_k|^{T-1} x_i(h), \quad \forall i \in I, \forall h \in \tilde{\mathcal{E}}_i. \tag{165}$$

- The set of constraints (119) of **MILP-n agents**, that is

$$y_i(\varphi(h)) - \frac{1}{|O_i|^{T-1}} \sum_{j \in \mathcal{E}} \mathcal{R}(\alpha, \langle h, j_{-i} \rangle) z(j) \quad = \quad w_i(h), \quad \forall h \in \mathcal{E}_i$$

must be replaced by

$$y_i(\varphi(h)) - \frac{1}{|\tilde{O}_i|^{T-1}} \sum_{j \in \tilde{\mathcal{E}}} \mathcal{R}(\alpha, \langle h, j_{-i} \rangle) z(j) \quad = \quad w_i(h), \quad \forall h \in \tilde{\mathcal{E}}_i. \tag{166}$$

Appendix E. Example using MA-Tiger

All these example are derived using the Decentralized Tiger Problem (MA-Tiger) described in Section 2.2. We have two agents, with 3 actions (a_l, a_r, a_o) and 2 observations (o_l, o_r). We will only consider problem with an horizon of 2.

There are 18 ($3^2 \times 2$) terminal histories for an agent: $a_o.o_l.a_o$, $a_o.o_l.a_l$, $a_o.o_l.a_r$, $a_o.o_r.a_o$, $a_o.o_r.a_l$, $a_o.o_r.a_r$, $a_l.o_l.a_o$, $a_l.o_l.a_l$, $a_l.o_l.a_r$, $a_l.o_r.a_o$, $a_l.o_r.a_l$, $a_l.o_r.a_r$, $a_r.o_l.a_o$, $a_r.o_l.a_l$, $a_r.o_l.a_r$, $a_r.o_r.a_o$, $a_r.o_r.a_l$, $a_r.o_r.a_r$.

And thus 324 ($18^2 = 3^{2\times2} \times 2^2$) joint histories for the agents: $\langle a_o.o_l.a_o, a_o.o_l.a_o \rangle$,$\langle a_o.o_l.a_o, a_o.o_l.a_l \rangle$, $\langle a_o.o_l.a_o, a_o.o_l.a_r \rangle$, $\cdots$, $\langle a_r.o_r.a_r, a_r.o_r.a_r \rangle$.

E.1 Policy Constraints

The policy constraints with horizon 2 for one agent in the MA-Tiger problem would be: Variables: x for every history

$$\begin{aligned}
x(a_o) + x(a_l) + x(a_r) &= 0 \\
-x(a_o) + x(a_o.o_l.a_o) + x(a_o.o_l.a_l) + x(a_o.o_l.a_r) &= 0 \\
-x(a_o) + x(a_o.o_r.a_o) + x(a_o.o_r.a_l) + x(a_o.o_r.a_r) &= 0 \\
-x(a_l) + x(a_l.o_l.a_o) + x(a_l.o_l.a_l) + x(a_l.o_l.a_r) &= 0 \\
-x(a_l) + x(a_l.o_r.a_o) + x(a_l.o_r.a_l) + x(a_l.o_r.a_r) &= 0 \\
-x(a_r) + x(a_r.o_l.a_o) + x(a_r.o_l.a_l) + x(a_r.o_l.a_r) &= 0 \\
-x(a_r) + x(a_r.o_r.a_o) + x(a_r.o_r.a_l) + x(a_r.o_r.a_r) &= 0
\end{aligned}$$

$$
\begin{array}{lll}
x(a_o) \geq 0 & x(a_l) \geq 0 & x(a_r) \geq 0 \\
x(a_o.o_l.a_o) \geq 0 & x(a_o.o_l.a_l) \geq 0 & x(a_o.o_l.a_r) \geq 0 \\
x(a_o.o_r.a_o) \geq 0 & x(a_o.o_r.a_l) \geq 0 & x(a_o.o_r.a_r) \geq 0 \\
x(a_l.o_l.a_o) \geq 0 & x(a_l.o_l.a_l) \geq 0 & x(a_l.o_l.a_r) \geq 0 \\
x(a_l.o_r.a_o) \geq 0 & x(a_l.o_r.a_l) \geq 0 & x(a_l.o_r.a_r) \geq 0 \\
x(a_r.o_l.a_o) \geq 0 & x(a_r.o_l.a_l) \geq 0 & x(a_r.o_l.a_r) \geq 0 \\
x(a_r.o_r.a_o) \geq 0 & x(a_r.o_r.a_l) \geq 0 & x(a_r.o_r.a_r) \geq 0
\end{array}
$$

E.2 Non-Linear Program for MA-Tiger

The Non-Linear Program for finding an optimal sequence-form policy for the MA-Tiger with horizon 2 would be:

Variables: x_i for every history for each agent

$$
\begin{array}{lll}
\text{Maximize} & & \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_o \rangle) x_1(a_o.o_l.a_o) x_2(a_o.o_l.a_o) \\
& + & \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_l \rangle) x_1(a_o.o_l.a_o) x_2(a_o.o_l.a_l) \\
& + & \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_r \rangle) x_1(a_o.o_l.a_o) x_2(a_o.o_l.a_r) \\
& + & \cdots
\end{array}
$$

subject to:

$$
\begin{aligned}
x_1(a_o) + x_1(a_l) + x_1(a_r) &= 0 \\
-x_1(a_o) + x_1(a_o.o_l.a_o) + x_1(a_o.o_l.a_l) + x_1(a_o.o_l.a_r) &= 0 \\
-x_1(a_o) + x_1(a_o.o_r.a_o) + x_1(a_o.o_r.a_l) + x_1(a_o.o_r.a_r) &= 0 \\
-x_1(a_l) + x_1(a_l.o_l.a_o) + x_1(a_l.o_l.a_l) + x_1(a_l.o_l.a_r) &= 0 \\
-x_1(a_l) + x_1(a_l.o_r.a_o) + x_1(a_l.o_r.a_l) + x_1(a_l.o_r.a_r) &= 0 \\
-x_1(a_r) + x_1(a_r.o_l.a_o) + x_1(a_r.o_l.a_l) + x_1(a_r.o_l.a_r) &= 0 \\
-x_1(a_r) + x_1(a_r.o_r.a_o) + x_1(a_r.o_r.a_l) + x_1(a_r.o_r.a_r) &= 0
\end{aligned}
$$

$$
\begin{aligned}
x_2(a_o) + x_2(a_l) + x_2(a_r) &= 0 \\
-x_2(a_o) + x_2(a_o.o_l.a_o) + x_2(a_o.o_l.a_l) + x_2(a_o.o_l.a_r) &= 0 \\
-x_2(a_o) + x_2(a_o.o_r.a_o) + x_2(a_o.o_r.a_l) + x_2(a_o.o_r.a_r) &= 0 \\
-x_2(a_l) + x_2(a_l.o_l.a_o) + x_2(a_l.o_l.a_l) + x_2(a_l.o_l.a_r) &= 0 \\
-x_2(a_l) + x_2(a_l.o_r.a_o) + x_2(a_l.o_r.a_l) + x_2(a_l.o_r.a_r) &= 0 \\
-x_2(a_r) + x_2(a_r.o_l.a_o) + x_2(a_r.o_l.a_l) + x_2(a_r.o_l.a_r) &= 0 \\
-x_2(a_r) + x_2(a_r.o_r.a_o) + x_2(a_r.o_r.a_l) + x_2(a_r.o_r.a_r) &= 0
\end{aligned}
$$

$$
\begin{array}{ccc}
x_1(a_o) \geq 0 & x_1(a_l) \geq 0 & x_1(a_r) \geq 0 \\
x_1(a_o.o_l.a_o) \geq 0 & x_1(a_o.o_l.a_l) \geq 0 & x_1(a_o.o_l.a_r) \geq 0 \\
x_1(a_o.o_r.a_o) \geq 0 & x_1(a_o.o_r.a_l) \geq 0 & x_1(a_o.o_r.a_r) \geq 0 \\
x_1(a_l.o_l.a_o) \geq 0 & x_1(a_l.o_l.a_l) \geq 0 & x_1(a_l.o_l.a_r) \geq 0 \\
x_1(a_l.o_r.a_o) \geq 0 & x_1(a_l.o_r.a_l) \geq 0 & x_1(a_l.o_r.a_r) \geq 0 \\
x_1(a_r.o_l.a_o) \geq 0 & x_1(a_r.o_l.a_l) \geq 0 & x_1(a_r.o_l.a_r) \geq 0 \\
x_1(a_r.o_r.a_o) \geq 0 & x_1(a_r.o_r.a_l) \geq 0 & x_1(a_r.o_r.a_r) \geq 0
\end{array}
$$

$$
\begin{array}{ccc}
x_2(a_o) \geq 0 & x_2(a_l) \geq 0 & x_2(a_r) \geq 0 \\
x_2(a_o.o_l.a_o) \geq 0 & x_2(a_o.o_l.a_l) \geq 0 & x_2(a_o.o_l.a_r) \geq 0 \\
x_2(a_o.o_r.a_o) \geq 0 & x_2(a_o.o_r.a_l) \geq 0 & x_2(a_o.o_r.a_r) \geq 0 \\
x_2(a_l.o_l.a_o) \geq 0 & x_2(a_l.o_l.a_l) \geq 0 & x_2(a_l.o_l.a_r) \geq 0 \\
x_2(a_l.o_r.a_o) \geq 0 & x_2(a_l.o_r.a_l) \geq 0 & x_2(a_l.o_r.a_r) \geq 0 \\
x_2(a_r.o_l.a_o) \geq 0 & x_2(a_r.o_l.a_l) \geq 0 & x_2(a_r.o_l.a_r) \geq 0 \\
x_2(a_r.o_r.a_o) \geq 0 & x_2(a_r.o_r.a_l) \geq 0 & x_2(a_r.o_r.a_r) \geq 0
\end{array}
$$

E.3 MILP for MA-Tiger

The **MILP** with horizon 2 for the agents in the MA-Tiger problem would be:
Variables:
$x_i(h)$ for every history of agent i
$z(j)$ for every terminal joint history

$$
\begin{array}{lcl}
\text{Maximize} & & \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_o \rangle) z(\langle a_o.o_l.a_o, a_o.o_l.a_o \rangle) \\
& + & \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_l \rangle) z(\langle a_o.o_l.a_o, a_o.o_l.a_l \rangle) \\
& + & \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_r \rangle) z(\langle a_o.o_l.a_o, a_o.o_l.a_r \rangle) \\
& + & \cdots
\end{array}
$$

subject to:

$$\begin{aligned}
x_1(a_o) + x_1(a_l) + x_1(a_r) &= 0 \\
-x_1(a_o) + x_1(a_o.o_l.a_o) + x_1(a_o.o_l.a_l) + x_1(a_o.o_l.a_r) &= 0 \\
-x_1(a_o) + x_1(a_o.o_r.a_o) + x_1(a_o.o_r.a_l) + x_1(a_o.o_r.a_r) &= 0 \\
&\cdots \\
x_2(a_o) + x_2(a_l) + x_2(a_r) &= 0 \\
-x_2(a_o) + x_2(a_o.o_l.a_o) + x_2(a_o.o_l.a_l) + x_2(a_o.o_l.a_r) &= 0 \\
-x_2(a_o) + x_2(a_o.o_r.a_o) + x_2(a_o.o_r.a_l) + x_2(a_o.o_r.a_r) &= 0 \\
&\cdots \\
z(\langle a_o.o_l.a_o, a_o.o_l.a_o\rangle) + z(\langle a_o.o_l.a_o, a_o.o_l.a_l\rangle) + z(\langle a_o.o_l.a_o, a_o.o_l.a_r\rangle) &= 2 \times x_1(a_o.o_l.a_o) \\
z(\langle a_o.o_l.a_o, a_o.o_l.a_o\rangle) + z(\langle a_o.o_l.a_l, a_o.o_l.a_o\rangle) + z(\langle a_o.o_l.a_r, a_o.o_l.a_o\rangle) &= 2 \times x_2(a_o.o_l.a_o) \\
z(\langle a_o.o_l.a_l, a_o.o_l.a_o\rangle) + z(\langle a_o.o_l.a_l, a_o.o_l.a_l\rangle) + z(\langle a_o.o_l.a_l, a_o.o_l.a_r\rangle) &= 2 \times x_1(a_o.o_l.a_l) \\
z(\langle a_o.o_l.a_o, a_o.o_l.a_l\rangle) + z(\langle a_o.o_l.a_l, a_o.o_l.a_l\rangle) + z(\langle a_o.o_l.a_r, a_o.o_l.a_l\rangle) &= 2 \times x_2(a_o.o_l.a_l) \\
&\cdots
\end{aligned}$$

$$\begin{array}{lll}
x_1(a_o) \geq 0 & x_1(a_l) \geq 0 & x_1(a_r) \geq 0 \\
x_1(a_o.o_l.a_o) \in \{0,1\} & x_1(a_o.o_l.a_l) \in \{0,1\} & x_1(a_o.o_l.a_r) \in \{0,1\} \\
x_1(a_o.o_r.a_o) \in \{0,1\} & x_1(a_o.o_r.a_l) \in \{0,1\} & x_1(a_o.o_r.a_r) \in \{0,1\} \\
\cdots & & \\
x_2(a_o) \geq 0 & x_2(a_l) \geq 0 & x_2(a_r) \geq 0 \\
x_2(a_o.o_l.a_o) \in \{0,1\} & x_2(a_o.o_l.a_l) \in \{0,1\} & x_2(a_o.o_l.a_r) \in \{0,1\} \\
x_2(a_o.o_r.a_o) \in \{0,1\} & x_2(a_o.o_r.a_l) \in \{0,1\} & x_2(a_o.o_r.a_r) \in \{0,1\} \\
\cdots & & \\
z(\langle a_o.o_l.a_o, a_o.o_l.a_o\rangle) \in \{0,1\} & z(\langle a_o.o_l.a_o, a_o.o_l.a_l\rangle) \in \{0,1\} & z(\langle a_o.o_l.a_o, a_o.o_l.a_r\rangle) \in \{0,1\} \\
z(\langle a_o.o_l.a_l, a_o.o_l.a_o\rangle) \in \{0,1\} & z(\langle a_o.o_l.a_l, a_o.o_l.a_l\rangle) \in \{0,1\} & z(\langle a_o.o_l.a_l, a_o.o_l.a_r\rangle) \in \{0,1\} \\
\cdots & &
\end{array}$$

E.4 MILP-2 Agents for MA-Tiger

The **MILP-2 agents** with horizon 2 for the agents in the MA-Tiger problem would be:
Variables:
$x_i(h)$, $w_i(h)$ and $b_i(h)$ for every history of agent i
$y_i(\varphi)$) for each agent and for every information set

$$\text{Maximize} \quad y_1(\varnothing)$$

subject to:

$$
\begin{aligned}
x_1(a_o) + x_1(a_l) + x_1(a_r) &= 0 \\
-x_1(a_o) + x_1(a_o.o_l.a_o) + x_1(a_o.o_l.a_l) + x_1(a_o.o_l.a_r) &= 0 \\
-x_1(a_o) + x_1(a_o.o_r.a_o) + x_1(a_o.o_r.a_l) + x_1(a_o.o_r.a_r) &= 0 \\
&\cdots \\
x_2(a_o) + x_2(a_l) + x_2(a_r) &= 0 \\
-x_2(a_o) + x_2(a_o.o_l.a_o) + x_2(a_o.o_l.a_l) + x_2(a_o.o_l.a_r) &= 0 \\
-x_2(a_o) + x_2(a_o.o_r.a_o) + x_2(a_o.o_r.a_l) + x_2(a_o.o_r.a_r) &= 0 \\
&\cdots \\
y_1(\varnothing) - y_1(a_o.o_l) - y_1(a_o.o_r) &= w_1(a_o) \\
y_1(\varnothing) - y_1(a_l.o_l) - y_1(a_l.o_r) &= w_1(a_l) \\
y_1(\varnothing) - y_1(a_r.o_l) - y_1(a_r.o_r) &= w_1(a_r) \\
y_2(\varnothing) - y_2(a_o.o_l) - y_2(a_o.o_r) &= w_2(a_o) \\
y_2(\varnothing) - y_2(a_l.o_l) - y_2(a_l.o_r) &= w_2(a_l) \\
y_2(\varnothing) - y_2(a_r.o_l) - y_2(a_r.o_r) &= w_2(a_r)
\end{aligned}
$$

$$
\begin{aligned}
y_1(a_o.o_l) - \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_o\rangle) x_2(a_o.o_l.a_o) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_l\rangle) x_2(a_o.o_l.a_l) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_r\rangle) x_2(a_o.o_l.a_r) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_l.o_l.a_o\rangle) x_2(a_l.o_l.a_o) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_l.o_l.a_l\rangle) x_2(a_l.o_l.a_l) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_l.o_l.a_r\rangle) x_2(a_l.o_l.a_r) \\
\cdots &= w_1(a_o.o_l.a_o)
\end{aligned}
$$

$$
\begin{aligned}
y_1(a_o.o_l) - \mathcal{R}(\alpha, \langle a_o.o_l.a_l, a_o.o_l.a_o\rangle) x_2(a_o.o_l.a_o) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_l, a_o.o_l.a_l\rangle) x_2(a_o.o_l.a_l) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_l, a_o.o_l.a_r\rangle) x_2(a_o.o_l.a_r) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_l, a_l.o_l.a_o\rangle) x_2(a_l.o_l.a_o) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_l, a_l.o_l.a_l\rangle) x_2(a_l.o_l.a_l) \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_l, a_l.o_l.a_r\rangle) x_2(a_l.o_l.a_r) \\
\cdots &= w_1(a_o.o_l.a_l)
\end{aligned}
$$

$$
\begin{aligned}
&\cdots \\
y_1(a_r.o_r) - \mathcal{R}(\alpha, \langle a_r.o_r.a_r, a_o.o_l.a_o\rangle)x_2(a_o.o_l.a_o) & \\
-\mathcal{R}(\alpha, \langle a_r.o_r.a_r, a_o.o_l.a_l\rangle)x_2(a_o.o_l.a_l) & \\
-\mathcal{R}(\alpha, \langle a_r.o_r.a_r, a_o.o_l.a_r\rangle)x_2(a_o.o_l.a_r) & \\
-\mathcal{R}(\alpha, \langle a_r.o_r.a_r, a_l.o_l.a_o\rangle)x_2(a_l.o_l.a_o) & \\
-\mathcal{R}(\alpha, \langle a_r.o_r.a_r, a_l.o_l.a_l\rangle)x_2(a_l.o_l.a_l) & \\
-\mathcal{R}(\alpha, \langle a_r.o_r.a_r, a_l.o_l.a_r\rangle)x_2(a_l.o_l.a_r) & \\
\cdots &= w_1(a_r.o_r.a_r)
\end{aligned}
$$

$$
\begin{aligned}
y_2(a_o.o_l) - \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_o\rangle)x_1(a_o.o_l.a_o) & \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_l, a_o.o_l.a_o\rangle)x_1(a_o.o_l.a_l) & \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_r, a_o.o_l.a_o\rangle)x_1(a_o.o_l.a_r) & \\
-\mathcal{R}(\alpha, \langle a_l.o_l.a_o, a_o.o_l.a_o\rangle)x_1(a_l.o_l.a_o) & \\
-\mathcal{R}(\alpha, \langle a_l.o_l.a_l, a_o.o_l.a_o\rangle)x_1(a_l.o_l.a_l) & \\
-\mathcal{R}(\alpha, \langle a_l.o_l.a_r, a_o.o_l.a_o\rangle)x_1(a_l.o_l.a_r) & \\
\cdots &= w_2(a_o.o_l.a_o)
\end{aligned}
$$

$$
\begin{aligned}
y_2(a_o.o_l) - \mathcal{R}(\alpha, \langle a_o.o_l.a_o, a_o.o_l.a_l\rangle)x_1(a_o.o_l.a_o) & \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_l, a_o.o_l.a_l\rangle)x_1(a_o.o_l.a_l) & \\
-\mathcal{R}(\alpha, \langle a_o.o_l.a_r, a_o.o_l.a_l\rangle)x_1(a_o.o_l.a_r) & \\
-\mathcal{R}(\alpha, \langle a_l.o_l.a_o, a_o.o_l.a_l\rangle)x_1(a_l.o_l.a_o) & \\
-\mathcal{R}(\alpha, \langle a_l.o_l.a_l, a_o.o_l.a_l\rangle)x_1(a_l.o_l.a_l) & \\
-\mathcal{R}(\alpha, \langle a_l.o_l.a_r, a_o.o_l.a_l\rangle)x_1(a_l.o_l.a_r) & \\
\cdots &= w_2(a_o.o_l.a_l) \\
\cdots &
\end{aligned}
$$

$$
\begin{aligned}
x_1(a_o) \leq 1 - b_1(a_o) \qquad & x_1(a_l) \leq 1 - b_1(a_l) \\
x_1(a_r) \leq 1 - b_1(a_r) \qquad & x_1(a_o.o_l.a_o) \leq 1 - b_1(a_o.o_l.a_o) \\
x_1(a_o.o_l.a_l) \leq 1 - b_1(a_o.o_l.a_l) \qquad & x_1(a_o.o_l.a_r) \leq 1 - b_1(a_o.o_l.a_r) \\
\cdots &
\end{aligned}
$$

$$
\begin{aligned}
w_1(a_o) \leq \mathcal{U}_1(a_o)b_1(a_o) \qquad & w_1(a_l) \leq \mathcal{U}_1(a_l)b_1(a_l) \\
w_1(a_r) \leq \mathcal{U}_1(a_r)b_1(a_r) \qquad & w_1(a_o.o_l.a_o) \leq \mathcal{U}_1(a_o.o_l.a_o)b_1(a_o.o_l.a_o) \\
w_1(a_o.o_l.a_l) \leq \mathcal{U}_1(a_o.o_l.a_l)b_1(a_o.o_l.a_l) \qquad & w_1(a_o.o_l.a_r) \leq \mathcal{U}_1(a_o.o_l.a_r)b_1(a_o.o_l.a_r) \\
\cdots &
\end{aligned}
$$

$$
\begin{array}{rcl}
x_1(a_o) \geq 0 & x_1(a_l) \geq 0 & x_1(a_r) \geq 0 \\
x_1(a_o.o_l.a_o) \geq 0 & x_1(a_o.o_l.a_l) \geq 0 & x_1(a_o.o_l.a_r) \geq 0 \\
\dots & & \\
w_1(a_o) \geq 0 & w_1(a_l) \geq 0 & w_1(a_r) \geq 0 \\
w_1(a_o.o_l.a_o) \geq 0 & w_1(a_o.o_l.a_l) \geq 0 & w_1(a_o.o_l.a_r) \geq 0 \\
\dots & & \\
b_1(a_o) \in \{0,1\} & b_1(a_l) \in \{0,1\} & b_1(a_r) \in \{0,1\} \\
b_1(a_o.o_l.a_o) \in \{0,1\} & b_1(a_o.o_l.a_l) \in \{0,1\} & b_1(a_o.o_l.a_r) \in \{0,1\} \\
\dots & & \\
y_1(\varnothing) \in (-\infty, +\infty) & & \\
y_1(a_o.o_l) \in (-\infty, +\infty) & y_1(a_o.o_r) \in (-\infty, +\infty) & \\
\dots & &
\end{array}
$$

... and the same for agent 2

References

Amato, C., Bernstein, D. S., & Zilberstein, S. (2007a). Optimizing memory-bounded controllers for decentralized POMDPs. In *Proc. of the Twenty-Third Conf. on Uncertainty in Artificial Intelligence (UAI-07)*.

Amato, C., Bernstein, D. S., & Zilberstein, S. (2007b). Solving POMDPs using quadratically constrained linear programs. In *Proc. of the Twentieth Int. Joint Conf. on Artificial Intelligence (IJCAI'07)*.

Amato, C., Carlin, A., & Zilberstein, S. (2007c). Bounded dynamic programming for decentralized POMDPs. In *Proc. of the Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM) in AAMAS'07*.

Amato, C., Dibangoye, J., & Zilberstein, S. (2009). Incremental policy generation for finite-horizon DEC-POMDPs. In *Proc. of the Nineteenth Int. Conf. on Automated Planning and Scheduling (ICAPS-09)*.

Anderson, B., & Moore, J. (1980). Time-varying feedback laws for decentralized control. *Nineteenth IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*, *19(1)*, 519–524.

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. (2004). Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, *22*, 423–455.

Bellman, R. (1957). *Dynamic programming.* Princeton University Press, Princeton, New-Jersey.

Bernstein, D., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, *27*(4), 819–840.

Bernstein, D. S., Hansen, E. A., & Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. In *Proc. of the Nineteenth Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 1287–1292.

Boularias, A., & Chaib-draa, B. (2008). Exact dynamic programming for decentralized pomdps with lossless policy compression. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS'08)*.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK '96), De Zeeuwse Stromen, Nederlands*.

Buffet, O., Dutech, A., & Charpillet, F. (2007). Shaping multi-agent systems with gradient reinforcement learning. *Autonomous Agent and Multi-Agent System Journal (AAMASJ)*, *15*(2), 197–220.

Cassandra, A., Kaelbling, L., & Littman, M. (1994). Acting optimally in partially observable stochastic domains. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI)*.

Chadès, I., Scherrer, B., & Charpillet, F. (2002). A heuristic approach for solving decentralized-POMDP: assessment on the pursuit problem. In *Proc. of the 2002 ACM Symposium on Applied Computing*, pp. 57–62.

Cornuéjols, G. (2008). Valid inequalities for mixed integer linear programs. *Mathematical Programming B*, *112*, 3–44.

Dantzig, G. B. (1960). On the significance of solving linear programming problems with some integer variables. *Econometrica*, *28(1)*, 30–44.

d'Epenoux, F. (1963). A probabilistic production and inventory problem. *Management Science*, *10(1)*, 98–108.

Diwekar, U. (2008). *Introduction to Applied Optimization* (2 edition). Springer.

Drenick, R. (1992). Multilinear programming: Duality theories. *Journal of Optimization Theory and Applications*, *72(3)*, 459–486.

Fletcher, R. (1987). *Practical Methods of Optimization*. John Wiley & Sons, New York.

Ghavamzadeh, M., & Mahadevan, S. (2004). Learning to communicate and act in cooperative multiagent systems using hierarchical reinforcement learning. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*.

Govindan, S., & Wilson, R. (2001). A global newton method to compute Nash equilibria. *Journal of Economic Theory*, *110*, 65–86.

Hansen, E., Bernstein, D., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proc. of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*.

Horst, R., & Tuy, H. (2003). *Global Optimization: Deterministic Approaches* (3rd edition). Springer.

James, M., & Singh, S. (2004). Learning and discovery of predictive state representations in dynamical systems with reset. In *Proc. of the Twenty-first Int. Conf. of Machine Learning (ICML'04)*.

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Koller, D., Megiddo, N., & von Stengel, B. (1994). Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC '94)*, pp. 750–759.

Koller, D., & Megiddo, N. (1996). Finding mixed strategies with small supports in extensive form games. *International Journal of Game Theory*, *25(1)*, 73–92.

Lemke, C. (1965). Bimatrix Equilibrium Points and Mathematical Programming. *Management Science, 11(7)*, 681–689.

Luenberger, D. (1984). *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Reading, Massachussetts.

McCracken, P., & Bowling, M. H. (2005). Online discovery and learning of predictive state representations. In *Advances in Neural Information Processing Systems 18 (NIPS'05).*

Nair, R., Tambe, M., Yokoo, M., Pynadath, D., & Marsella, S. (2003). Taming decentralized POMDPs: towards efficient policy computation for multiagent setting. In *Proc. of Int. Joint Conference on Artificial Intelligence, IJCAI'03.*

Oliehoek, F., Spaan, M., & Vlassis, N. (2008). Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research (JAIR), 32*, 289–353.

Oliehoek, F., Whiteson, S., & Spaan, M. (2009). Lossless clustering of histories in decentralized POMDPs. In *Proc. of The International Joint Conference on Autonomous Agents and Multi Agent Systems*, pp. 577–584.

Osborne, M. J., & Rubinstein, A. (1994). *A Course in Game Theory.* The MIT Press, Cambridge, Mass.

Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications.

Papadimitriou, C. H., & Tsitsiklis, J. (1987). The Complexity Of Markov Decision Processes. *Mathematics of Operations Research, 12 (3)*, 441 – 450.

Parsons, S., & Wooldridge, M. (2002). Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems (JAAMAS), 5(3)*, 243–254.

Petrik, M., & Zilberstein, S. (2007). Average-reward decentralized Markov decision processes. In *Proc. of the Twentieth Int. Joint Conf. on Artificial Intelligence (IJCAI 2007).*

Petrik, M., & Zilberstein, S. (2009). A bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research (JAIR), 35*, 235–274.

Puterman, M. (1994). *Markov Decision Processes: discrete stochastic dynamic programming.* John Wiley & Sons, Inc. New York, NY.

Pynadath, D., & Tambe, M. (2002). The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories And Models. *Journal of Artificial Intelligence Research, 16*, 389–423.

Radner, R. (1959). The application of linear programming to team decision problems. *Management Science, 5*, 143–150.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A modern approach.* Prentice Hall.

Sandholm, T. (1999). *Multiagent systems*, chap. Distributed rational decision making, pp. 201–258. The MIT Press. Ed. by G. Weiss.

Sandholm, T., Gilpin, A., & Conitzer, V. (2005). Mixed-integer programming methods for finding nash equilibria. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*.

Scherrer, B., & Charpillet, F. (2002). Cooperative co-learning: A model based approach for solving multi agent reinforcement problems. In *Proc. of the IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI'02)*.

Seuken, S., & Zilberstein, S. (2007). Memory-bounded dynamic programming for DEC-POMDPs. In *Proc. of the Twentieth Int. Joint Conf. on Artificial Intelligence (IJCAI'07)*.

Singh, S., Jaakkola, T., & Jordan, M. (1994). Learning without state estimation in partially observable markovian decision processes.. In *Proceedings of the Eleventh International Conference on Machine Learning*.

Singh, S., Littman, M., Jong, N., Pardoe, D., & Stone, P. (2003). Learning predictive state representations. In *Proc. of the Twentieth Int. Conf. of Machine Learning (ICML'03)*.

Szer, D., & Charpillet, F. (2006). Point-based Dynamic Programming for DEC-POMDPs. In *Proc. of the Twenty-First National Conf. on Artificial Intelligence (AAAI 2006)*.

Szer, D., Charpillet, F., & Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proc. of the Twenty-First Conf. on Uncertainty in Artificial Intelligence (UAI'05)*, pp. 576– 583.

Thomas, V., Bourjot, C., & Chevrier, V. (2004). Interac-DEC-MDP : Towards the use of interactions in DEC-MDP. In *Proc. of the Third Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'04), New York, USA*, pp. 1450–1451.

Vanderbei, R. J. (2008). *Linear Programming: Foundations and Extensions* (3rd edition). Springer.

von Stengel, B. (2002). *Handbook of Game Theory*, Vol. 3, chap. 45-"Computing equilibria for two-person games", pp. 1723–1759. North-Holland, Amsterdam.

Wu, J., & Durfee, E. H. (2006). Mixed-integer linear programming for transition-independent decentralized MDPs. In *Proc. of the fifth Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'06)*, pp. 1058–1060 New York, NY, USA. ACM.

Xuan, P., Lesser, V., & Zilberstein, S. (2000). Communication in multi-agent Markov decision processes. In *Proc. of ICMAS Workshop on Game Theoretic and Decision Theoretics Agents* Boston, MA.

Rubrique

Coordination par les Jeux Stochastiques

Apprentissage par Renforcement et Théorie des Jeux pour la coordination de Systèmes Multi-Agents

Alain Dutech — Raghav Aras — François Charpillet

LORIA - INRIA
615, rue du jardin botanique
Villers les Nancy
France
{dutech,aras,charp}@loria.fr

RÉSUMÉ. Nous présentons les principaux algorithmes d'apprentissage par renforcement visant à coordonner des systèmes multi-agents en s'appuiant sur des outils et des notions tirés de la Théorie des Jeux. Les limitations des ces approches sont évoquées et discutées afin d'essayer de dégager des problématiques prometteuse pour ce champs de recherche. Nous y discutons en particulier la pertinence des équilibres de Nash et des jeux à information partielle.

ABSTRACT. This article presents the main reinforcement learning algorithms that aim at coordinating multi-agent systems by using tools and formalisms borrowed from Game Theory. Limits of these approaches are studied and discussed in order to draw some promising lines of research for that particular field. We argue more deeply around the central notions of Nash equilibrium and games with imperfect monitoring.

MOTS-CLÉS : Systèmes Multi-Agents, Coordination, Théorie des Jeux, Apprentissage par Renforcement

KEYWORDS : Multi-Agent Systems, Coordination, Game Theory, Reinforcement Learning

1. Introduction

Les méthodes d'apprentissage par renforcement (AR) [16] et leur formalisation par les processus décisionnels de Markov (MDP[1]) [15] ont pris une place importante dans les travaux de recherche en intelligence artificielle. Plus récemment, leur utilisation a été élargie aux systèmes multi-agents (SMA), en tant que méthode de construction automatique mais aussi en tant que formalisme. Cet élargissement a conduit assez naturellement à utiliser et à s'inspirer du bagage mathématique de la Théorie des Jeux (TdJ) [14] pour étendre les outils et formalismes classiques de l'AR.

Nous allons plus précisément nous attacher au problème de la coordination et de la coopération rationnelle d'agents. La question posée est alors de savoir si les algorithmes qui s'appuient sur la combinaison des formalismes des MDP et de la TdJ permettent effectivement à des agents de se coordonner, voire de coopérer.

Le but de cet article est en fait triple. Premièrement, nous voulons présenter de manière synthétique les travaux majeurs de ce courant de recherche qui allie AR et TdJ. Cette présentation essaiera aussi de dégager les principaux concepts mis en oeuvre en apprentissage par renforcement. Deuxièmement, nous voulons essayer de mettre en avant les problèmes et les limitations des travaux actuels. De cette discussion, nous ferons émerger les problématiques qui nous semblent les plus intéressantes et qui sont, ou seront, les points de focalisations des recherches futures. Nous discuterons notamment de la pertinence des équilibres de Nash et de l'intérêt de pouvoir traiter des problèmes d'information partielle. Enfin, nous proposerons des pistes, plus ou moins défrichées, pour avancer dans les directions qui nous paraissent les plus prometteuses.

Nous avons donc organisé cet article comme suit. La Section 2 présente le premier formalisme, au sens chronologique, alliant MDP et SMA. Outre la présentation des principes des MDP, cette partie explique comment le problème de coordination devient un problème lié à la TdJ. La Section 3 présente alors les principaux travaux s'appuyant à la fois sur la TdJ et les MDP pour résoudre ce problème de coordination, alors que la Section 4 s'y intéresse quant les agents n'ont que des informations partielles sur leur environnement. Notre point de vue sur les limitations et les problèmes les plus intéressants qui restent à résoudre est explicité en Section 5. Nous y ajoutons des propositions de direction de recherche plus ou moins concrètes en Sections 5.3 et 5.4 avant de conclure.

2. Rationalité multi-agents et coordination

Les travaux de Boutilier [2, 6] sur les processus décisionnels Markoviens *multi-agents* (MMDP) ont été parmi les premiers à poser le problème de la conception de systèmes multi-agents en terme de théorie de la décision.

Un MMDP est un tuple $\langle \mathcal{S}, N, \mathcal{A} = \{\mathcal{A}_i\}_{i \in N}, p, r \rangle$ où : $\mathcal{S}$ et N sont des ensembles finis d'états et d'agents ; $\mathcal{A}_i$ est un ensemble fini d'action pour l'agent i ; $p : \mathcal{S} \times \mathcal{A}_1 ... \times \mathcal{A}_n \times \mathcal{S} \rightarrow [0, 1]$ est une fonction de transitions stochastique entre les états et $r : \mathcal{S} \rightarrow \mathbb{R}$ est une fonction de récompense globale.

Dans ce modèle, le système étant dans un état s_t au temps t, les agents choisissent une action a_i, ce qui forme une action jointe $a = \{a_i\}$ et qui modifie l'état du système de manière stochastique : $\Pr(s_{t+1}|s_t, a) = p(s_t, a, s_{t+1})$. Les agents, reçoivent alors une

1. pour cet acronyme, comme pour d'autre par la suite, nous utiliserons la version anglaise qui est souvent plus parlante et plus connue

récompense $r_{t+1} = r(s_t, a)$. Le problème posé est de trouver les actions qui maximisent un critère fonction de la récompense reçue, classiquement $E[\sum_{t=0}^{\infty} \gamma^t r_t]$ où $\gamma \in [0, 1[$.

De manière analogue au cadre mono-agent largement étudié (voir [15, 16]), on formalise les décisions des agents sous la forme de politiques d'action $\pi_i : \mathcal{S} \rightarrow \Delta(\mathcal{A}_i)$ ($\Delta(.)$ dénotant une distribution). Pour chaque politique d'action jointe $\pi = \{\pi_i\}$, on peut associer une *fonction valeur* $Q(s, a)$ qui, pour chaque état, est l'espérance de sa valeur (au sens du critère précédent) si les agents choisissent d'abord l'action a avant de suivre la politique π. Dans ce cadre, on sait qu'il existe des politiques jointes optimales π^* dont la fonction valeur optimale Q^* vérifie l'équation de Bellman :

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s, a, s') \max_{a' \in \mathcal{A}} Q^*(s', a') \quad [1]$$

On a alors que $\pi^*(s) = \text{argmax}_{a \in \mathcal{A}} Q^*(s, a)$.

Chaque agent peut individuellement, en utilisant des algorithmes classiques [15, 16], trouver des politiques optimales. Néanmoins, s'il existe plusieurs politiques optimales, les agents *doivent* choisir, de manière indépendante, la même politique optimale : ils doivent se coordonner. Boutilier montre qu'apprendre à se coordonner revient à apprendre des équilibres dans des jeux à n joueurs.

3. MDP multi-agents et recherche d'équilibres

3.1. Q-learning Multi-agent générique

La forme générale des algorithmes d'apprentissage par renforcement pour apprendre des équilibres dans des jeux est la suivante : (1) les agents, dans un état s, choisissent leurs actions $a_1, ..., a_n$; (2) l'état du système est modifié en s' et les agents reçoivent une récompense $r_1, ..., r_n$; (3) les équations 2 et 3 permettent aux agents de ré-estimer leur politique optimale ($\alpha \in]0, 1[$ est un coefficent d'apprentissage) ; (4) on revient au point (1) avec $s = s'$

$$V_i(s') \quad \longleftarrow \quad f_i(Q_1(s', .), ..., Q_n(s', .)) \quad [2]$$

$$Q_i(s, a) \quad \longleftarrow \quad (1 - \alpha) Q_i(s, a) + \alpha[r_i + \gamma V_i(s')] \quad [3]$$

Les différents travaux que nous allons passer en revue se différencient principalement par la forme générale que peuvent prendre les fonctions récompenses et surtout par les fonctions f_i qui choisissent, à partir des informations courantes sur les matrices de récompenses de *tous les joueurs*, les équilibres à utiliser. On peut alors remarquer que, de manière générale, les agents *doivent* pouvoir observer les actions et les récompenses des autres agents pour maintenir cette connaissance sur les autres.

3.2. Déclinaisons de l'algorithme générique

Avec le *minimax-Q* [11], Littman s'est d'abord intéressé à des jeux à somme nulle à deux joueurs. Les équilibres y sont les politiques où chaque joueur cherche à maximiser ses gains en minimisant celui de l'autre. La fonction f_i est alors :

$$V_1(s) = \max_{\sigma_1 \in \Delta(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} Q_1(s, (\sigma_1, a_2)) = -V_2(s)$$

Néanmoins, en dépit de ses garanties de convergence vers des politiques optimales (avec des limitations forte sur la nature du jeux, voir [13]), le cadre applicatif de cet algorithme

le rend impropre à la recherche de coordination car, à cause de la forme des récompenses, les agents ont ici des but totalement opposés.

Si l'on s'intéresse au cadre plus complet des jeux à somme générale (les récompenses individuelles peuvent y être quelconque), une généralisation des travaux précédents à été proposée par Hu et Wellman [9] avec l'algorithme *Nash-Q*. Le principe est de s'appuyer sur les équilibres de Nash [14] pour coordonner les agents. La fonction de choix d'équilibre devient alors $V_i(s) \in \mathrm{NASH}_i(Q_1(s,.),...,Q_n(s,.))$ où NASH_i est l'ensemble des équilibres de Nash du jeux dans l'état s. Il apparaît rapidement que les jeux possédant plusieurs équilibres différents peuvent poser problème. De fait, comme établit dans [10], la convergence de cet algorithme n'est garantie que pour des jeux possédant des caractéristiques très particulières.

Pour s'affranchir un peu de la nécessité pour chaque agent de pouvoir observer les actions et les récompenses des autres afin de trouver les équilibres de Nash, Littman [12] a proposé une version plus spécialisée du *Nash-Q* appelée *Friend-or-Foe Q-Learning* (FoFQ). Si les joueurs doivent savoir à quelle forme de jeu ils sont confrontés, ils n'ont alors plus besoin de tout savoir sur les autres joueurs.

Dans l'algorithme *Correlated Q-Learning* (CorrQ) proposé par Greenwald et Hall [7] la fonction f_i de sélection d'équilibre ne cherche plus des équilibres de Nash (ou approchés) mais cherche des *équilibres corrélés* qui sont plus adéquats quand on veut faire coopérer des agents. Ce sont des équilibres sur les actions jointes (ce qui suppose une certaine confiance entre les agents) mais qui permettent de recevoir une récompense au moins égale au meilleur équilibre de Nash.

De manière pratique, la recherche d'équilibre μ (où $\mu(\pi)$ est la probabilité qu'une politique *déterministe* soit recommandée) se fait par programmation linéaire avec le système d'inégalité suivant :

$$\sum_{\pi_{-i} \in \Pi_{-i}} \mu(\pi)(r_i(\pi) - r_i(\pi_{-i}, \pi_i)) \quad \geq 0 \quad \forall i \in N, \forall \pi \in \Pi_i, \forall \pi_i \in \Pi_i \qquad [4]$$

Outre le fait qu'il peut exister plusieurs équilibre corrélés (et qu'il faut donc des heuristiques pour choisir parmi ces équilibres), l'algorithme nécessite aussi que chaque agent puisse observer les actions et les récompenses des autres agents.

3.3. Algorithme "Win or Learn Fast"

L'algorithme *WoLF* (pour "Win or Learn Fast") de Bowling et Veloso [3] sort du cadre générique de la Section 3.1. Cet algorithme s'inspire des algorithmes d'itération de la politique. Pour une politique *stochastique* donnée π_i, un joueur peut calculer une estimation de la valeur de cette politique. Après chaque interaction avec le jeu, le joueur peut utiliser cette estimation de la valeur pour améliorer sa politique actuelle en augmentant d'une quantité δ la probabilité que l'action "optimale" soit choisie à l'avenir. Le coefficient d'apprentissage δ dépend du fait que le joueur soit en train de gagner (δ sera alors faible) ou de perdre (δ plus grand). Plus précisément :

$$\delta \quad = \quad \begin{cases} \delta_w & \text{si } \sum_{a'_i} \pi_i(s,a'_i)Q(s,a'_i) > \sum_{a'_i} \bar{\pi}_i(s,a'_i)Q(s,a'_i) \\ \delta_l & \text{sinon} \end{cases} \qquad [5]$$

où $\delta_w < \delta_l$ et $\bar{\pi}_i$ est la politique moyenne suivie par l'agent i.

Dans le cas précis d'un jeu itéré à deux joueurs avec deux actions, si les deux joueurs utilisent l'algorithme *WoLF*, alors leurs politiques vont converger vers un équilibre de

Nash. La version théorique de l'algorithme nécessite que chaque agent connaisse sa fonction de récompense, la politique de l'autre joueur. En pratique, cet algorithme a été validé sur des jeux à deux joueurs plus complexes (plus d'actions, jeux stochastiques) où il a convergé vers des politiques qui étaient les meilleurs réponses possibles aux stratégies adverses.

4. Jeux avec information partielle

Le cas des jeux avec informations partielle, où les joueurs ne connaissent pas exactement l'état ni les actions et récompenses reçue par les autres joueurs, est notablement plus complexe. Du point de vue de l'apprentissage par renforcement, ces jeux se rapprochent des processus décisionnels Markoviens partiellement observables (POMDP, voir [4]).

4.1. Formalisme

Les jeux stochastiques partiellement observables (POSG) sont définis par $\langle \mathcal{S}, N, \mathcal{A} = \{\mathcal{A}_i\}, \{\Omega_i\}, p, r \rangle$. On a donc ajouté, pour chaque agent, un ensemble Ω_i d'observations et les fonctions de transitions p sont modifiées pour fournir, en plus de la probabilité de l'état suivant, la probabilité des observations :$p(s_t, a, s_{t+1}, o = (o_i)) = \Pr(s_{t+1}, o | s_t, a)$. Quant aux joueurs, ils ne connaissent pas l'état mais peuvent seulement l'observer.

4.2. Elagage itératif

Il est intéressant de noter que les travaux les plus avancés sur le sujet, ceux de Hansen, Bernstein et Zilberstein [8] ont vraiment rapproché le domaines de l'apprentissage par renforcement (plus particulièrement des POMDP) et de la théorie des jeux.

L'idée principale pour résoudre un POMDP est de se ramener à un cas totalement observable en ne considérant plus les états du processus mais des distributions de probabilités sur ces états. On parle alors d'état de croyance $b(s) = \Pr(s_t = s)$ qui sont définis sur $\Delta(\mathcal{S})$. La valeur d'une politique π est une fonction linéaire par morceau qui peut s'exprimer comme $V(b) = max_j \sum_{s \in \mathcal{S}} b(s) v_j(s)$ où $\mathcal{V} = \{v_1, ..., v_k\}$ est un ensemble de vecteurs. Ces vecteurs sont tous des combinaisons des vecteurs représentatifs des sous-politiques de π. En partant de politiques simples (une seule action), un calcul récursif permet alors de calculer la politique optimale. A chaque étape, un élagage des vecteurs inutiles permet d'éviter une explosion combinatoire du nombre de vecteurs [5].

Cette idée a été étendue aux POSG par Hansen et al. [8] (*DP-POSG*), mais avec une notion de croyance adaptée au cadre multi-agent. Ainsi, pour un joueur i, un état de croyance b_i est définit comme étant une distribution sur $\mathcal{S} \times \Pi_{-i}$ où Π_{-i} est l'ensemble des politiques possibles des autres agents. Ainsi, en une étape similaire au test de dominance utilisé dans les POMDP, en s'appuyant aussi sur la programmation linéaire, il est possible de mettre en oeuvre une étape d'élagage dans les POSG qui élimine les politiques déterministes π_i qui sont très faiblement dominées, c'est-à-dire où il existe une politique stochastique μ_i telle que :

$$V_i(s, (\mu_i, \pi_{-i})) \quad \geq \quad V_i(s, (\pi_i, \pi_{-i})),\ \forall s \in S,\ \forall \pi_{-i} \in \Pi_{-i} \qquad [6]$$

Il est alors prouvé qu'un tel algorithme directement inspiré de la programmation dynamique, met en oeuvre une méthode d'élimination itérative des politiques très faiblement dominées, méthode bien connue de la théorie des jeux [14]. On génère ainsi un ensemble de politique qui *peut* contenir tous les équilibres de Nash.

Algo	Connaît $\mathcal{S}$	Connaît $\mathcal{A}+r$	r restreint	Equ. Nash	Garanties Théoriques
minimaxQ	.	.	.	.	+
NashQ	.	.	+	.	$\sim$
FoFQ	.	+	.	.	+
WoLF	.	+	$\sim$	.	$\sim$
CorrQ	.	.	+	+	.
DP-POSG	+	+	+	.	$\sim$
BCE-Q	+	+	$\sim$	+	.

Tableau 1. *Comparaison des algorithmes évoqués (voir texte). Un '+' signifie que l'algorithme n'est pas dépendant de cette contrainte, un '$\sim$' que cette dépendance est forte mais pas absolue, et un '.' que l'algorithme dépend de cette contrainte.*

5. Discussion et propositions

La Table 5 résume les caractéristiques des algorithmes auxquelles nous allons nous attacher lors de cette discussion.

5.1. Agents omniscients

Les algorithmes qui s'appuient sur les MDP (voir Section 3) sont quasiment tous dépendants du fait que les agents doivent connaître l'état du jeu. Ils doivent aussi pouvoir observer les actions et les récompenses des autres joueurs. Ces contraintes sont difficilement compatibles avec le formalisme des systèmes multi-agents qui s'appuient sur la localité des agents, ce qui implique une connaissance partielle du système et des autres.

L'algorithme DP-POSG de Hansen et Bernstein est une réponse possible à ce problème, mais limitée par sa complexité et sa recherche d'équilibres de Nash, d'ailleurs peu efficaces(voir Section 5.3).

5.2. Coordination, Coopération et Nash

En fait, s'il apparaît évident que le fait de pouvoir traiter des jeux où les agents ont tous la même récompense est nécessaire pour construire des agents coopérants, ce n'est pas suffisant. En effet, on ne peut écarter le cas d'agents légèrement différents voulant tout de même coopérer localement et temporairement, et dans ce cas il faut pouvoir gérer des jeux où chaque agent a une fonction de récompense propre et indépendante des autres agents.

De plus, les équilibres de Nash peuvent être globalement *inefficaces* : il existe de nombreux jeux où l'équilibre de Nash est moins efficace que d'autres choix mais qui nécessitent soit des médiateurs, soit de la communication, soit des agents se faisant confiance pour être choisis par les agents. C'est justement ce que nous attendons d'agent devant coopérer. L'algorithme *Correlated-Q* répond en partie à ce problème, mais nécessite des agents omniscients.

5.3. Equilibres séquentiels dans les POSG

Dans l'algorithme *DP-POSG*, le fait d'éliminer des sous-politiques même très faiblement dominées, fait courir le risque d'éliminer des équilibres de Nash qui sont, eux, globaux. Ce n'est pas gênant si ces équilibres sont en fait irréalistes (voir [14]) mais rien

ne prouve que cela soit le cas. En fait, l'algorithme semblerait mieux adapté à la recherche d'*équilibres séquentiels* qui sont, aussi, des équilibres de sous-politiques.

Nous sommes donc en train de réfléchir à une adaptation de cet algorithme, notamment au travers d'heuristique pour déterminer quels sont les sous-jeux d'un jeu donné (notion rendue complexe par l'observabilité partielle de l'état), car il est alors possible d'utiliser des tests de dominance locaux dans ses sous-jeux sans pour autant éliminer d'équilibres séquentiels. Ce sont des travaux de recherche en cours.

5.4. Communication limitée

Pour une utilisation plus réaliste des jeux comme cadre formel de la coordination multi-agents, les algorithmes proposés ne doivent pas dépendre du fait que les agents puisse observer les actions et les récompenses des autres agents. La recherche de politiques s'appuyant sur la notion d'équilibre de Nash n'est pas non plus satisfaisante pour l'apprentissage de la coordination car les agents se limitent alors seulement à limiter leurs pertes, sans chercher à coopérer.

Nous avons donc proposé un formalisme d'apprentissage appelé *Best Compromise Equilibrium Q-Learning* (BCE-Q) [1] qui respecte ces contraintes dans le cadre général des POSG. Ce formalisme s'appuie sur une communication la plus limitée possible (pour ne pas alourdir plus que de raison le processus d'apprentissage) entre les agents pour mettre en oeuvre un recherche d'équilibre s'inspirant de méthodes des transferts de récompenses entre les agents, méthodes parfois mises en oeuvre dans les jeux de négociation.

Le principe général de cet algorithme est de modifier les récompenses reçues par les agents en fonctions des messages binaires (oui/non) envoyés par les agents et de la récompense données par l'environnement. La forme générale de cette récompense "virtuelle" est $\hat{r}_i = (x|m_{i.s}| + y|m_{i.r}| + z)r_i$ où x, y et z sont des paramètres (à choisir ou apprendre) et $|m_{i.s}$ (resp. $|m_{i.r}|$) est le nombre de messages envoyés (resp. reçus) par l'agent i. Cet algorithme a été testé expérimentalement avec des paramètres fixés manuellement pour des jeux stochastiques et permet de trouver des politiques plus performantes que celles trouvées par *Nash-Q*.

6. Conclusion

Nous avons essayé de faire un tour d'horizon à la fois complet et synthétique des recherches actuelles qui visent à allier la Théorie des Jeux et les Processus Décisionnels de Markov pour coordonner rationnellement des agents. Nombre de ces travaux se sont concentrés sur la possibilité d'apprendre des équilibres de Nash. Nous avons argumenté que, d'une part, ces algorithmes nécessitent des agents omniscients et donc peu réalistes ; d'autre part, les équilibres de Nash ne sont pas forcément adaptés au problèmes de coordination et coopération.

DP-POSG [8] est un des rares algorithmes qui s'attaque au problèmes d'agents ayant des informations limitées. Nous avons alors montré que cet algorithme souffrait de quelques défauts et suggérés des pistes de travail possible pour les corriger. Nous avons aussi proposé un algorithme [1] qui essaie de répondre aux deux critiques formulées plus haut en permettant aux agents une communication limitée et un transfert de récompense.

Ces travaux sont toujours en développement et montrent que les questions de communication et d'information partielles restent des voies de recherche importantes et actuelles.

7. Bibliographie

[1] ARAS, R., DUTECH, A., AND CHARPILLET, F. Cooperation in stochastic games through communication. In *Proc. of the fourth Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'05)* (Utrech, Nederlands, 2005).

[2] BOUTILIER, C. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK '96), De Zeeuwse Stromen, The Netherlands* (1996).

[3] BOWLING, M., AND VELOSO, M. Multiagent learning using a variable learning rate. *Artificial Intelligence 136* (2002), 215–250.

[4] CASSANDRA, A. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Department of Computer Science, Providence, RI, 1998.

[5] CASSANDRA, A., LITTMAN, M., AND ZHANG, N. Incremental pruning : A simple, fast, exact method for partially observable markov decision processes. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)* (1997).

[6] CLAUS, C., AND BOUTILIER, C. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI* (1998), pp. 746–752.

[7] GREENWALD, A., AND HALL, K. Correlated Q-learning. In *Proc. of the 20th Int. Conf. on Machine Learning (ICML)* (2003).

[8] HANSEN, E., BERNSTEIN, D., AND ZILBERSTEIN, S. Dynamic programming for partially observable stochastic games. In *Proc. of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)* (2004).

[9] HU, J., AND WELLMAN, M. Multiagent reinforcement learning : theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning, ICML-98* (1998), pp. 242–250.

[10] HU, J., AND WELLMAN, M. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research* (2003).

[11] LITTMAN, M. Markov games as a framework for multi-agent reinforcement learning. In *Proceddings of the eleventh Int. Conference on Machine Learning, San Fransisco, CA* (1994).

[12] LITTMAN, M. Friend-or-foe Q-learning in general-sum games. In *Proc. of the 18th Int. Conf. on Machine Learning (ICML)* (2001).

[13] LITTMAN, M., AND SZEPESVÁRI, C. A generalized reinforcement-learning model : Convergence and applications. In *Proc. of the Thirteenth Int. Conf. on Machine Learning (ICML'96)* (1996).

[14] MYERSON, R. *Game Theory : Analysis of Conflict*. Harvard University Press, 1991.

[15] PUTERMAN, M. *Markov Decision Processes : discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, NY, 1994.

[16] SUTTON, R., AND BARTO, G. *Reinforcement Learning*. Bradford Book, MIT Press, Cambridge, MA, 1998.

Shaping Multi-Agent Systems with Gradient Reinforcement Learning *

Olivier Buffet (olivier.buffet@laas.fr)
LAAS/CNRS
Groupe RIS
7 Avenue du Colonel Roche
31077 Toulouse CEDEX 4
France

Alain Dutech (alain.dutech@loria.fr) and
François Charpillet (francois.charpillet@loria.fr)
Loria - INRIA-Lorraine
Campus Scientifique - BP 239
54506 Vandœuvre-lès-Nancy CEDEX
France

November 1, 2006

Abstract. An original Reinforcement Learning (RL) methodology is proposed for the design of multi-agent systems. In the realistic setting of situated agents with local perception, the task of automatically building a coordinated system is of crucial importance. To that end, we design simple reactive agents in a decentralized way as independent learners. But to cope with the difficulties inherent to RL used in that framework, we have developed an incremental learning algorithm where agents face a sequence of progressively more complex tasks. We illustrate this general framework by computer experiments where agents *have* to coordinate to reach a global goal.

Keywords: Reinforcement Learning, Multi-Agent Systems, Partially Observable Markov Decision Processes, Shaping, Policy-Gradient

Abbreviations: RL – Reinforcement Learning; MAS – Multi-Agent System; MDP – Markov Decision Process; POMDP – Partially Observable Markov Decision Process

1. Introduction

The problem of automating the design of a Multi-Agent System (MAS) is at the heart of much research. It is an important issue mostly related to a crucial question: how to link the *global* description of a task with agents whose behaviours depend on a necessarily *partial* and *local* view of this task.

This design problem is all the more tricky as "reactive" and "cooperative" MAS are considered, since these systems rely on interactions

* This work has been conducted in part in NICTA's Canberra laboratory.

among—often many—agents in order to produce a complex collective behaviour relying in particular on self-organisation phenomena. The main difficulty comes from the fact that self-organisation is a process that is not well understood and not easy to control. Thus, the only solution for the designer is often to undergo a tedious task of tuning the parameters of the reactive behaviours so as to obtain the desired collective behaviour. On the other hand, the greatest advantage of these systems is the simplicity of the agents, which eases their design.

Reinforcement Learning (RL) [45, 6] is a common approach to decision-making under uncertainty. It is also an appealing tool to tackle the automation of MAS design [43, 38] in a decentralised manner. This is the approach adopted in the present paper, in which independent learners try to achieve a common goal. Our interest is to see each agent learn locally how to optimise a global performance. There are many advantages to this approach:

- Because we employ Reinforcement Learning, a simple scalar signal evaluating the system behaviour (the reward) is sufficient to learn. It is not necessary to have a teacher knowing the problem's solution beforehand.

- The decentralised framework often makes the task faced by each agent of a reactive MAS rather simple. In many cases, it is easier for each agent to learn its local reactive behaviour than to try learning the system's collective behaviour. This is a way to avoid an important pitfall of RL: a combinatorial explosion is all the more probable that the problem is complex.

Moreover, Reinforcement Learning methods are based on the mathematical formalism of *Markov Decision Processes* (MDP) that details the assumptions to be satisfied for an algorithm to converge. This mathematical framework also gives a formalism for the study of a MAS's collective behaviour.

The use of Reinforcement Learning in a decentralised fashion for Multi-Agent Systems causes some difficulties. Indeed, our approach is not in the precise framework of MDPs (because of the multi-agent partially observable setting), which leads to the loss of the usual guarantees that the algorithm converges to an optimal behaviour. This is due to the fact that each agent is constrained to a local and partial view of the system and, as a consequence, generally does not know the system's global state[1].

Common model-free learning algorithms are for example Q-learning and Sarsa [45]. Under the Markov assumption, they efficiently find a globally optimal deterministic policy (one such policy exists in this

case). Yet, because each of our learners is facing a non-Markovian problem (due to a decision made with insufficient information), we have to use direct policy search algorithms instead, looking for an—at least—*locally* optimal *stochastic* policy.

Based on such a classical RL technique to find stochastic policies, our proposition is then to make use of a *decentralised incremental* learning algorithm:

- The learning is *decentralised* because the group's behaviour evolves through the independent learning processes of each agent.
- By *incremental*, we mean that agents are progressively pitted against harder and harder tasks so as to progressively learn a more complex behaviour.

Another way in which our approach could be called incremental is that initially learning is performed with very few agents, so as to minimise coordination and cross-work actions, and then the resulting basic behaviours are exported to learning tasks with more and more agents. Eventually, behaviours can be further refined by learning in these more demanding environments. Thus, the originality of our approach is twofold: learning is decentralised and incremental.

Note: In a broader sense, incremental RL is generally referred to as *shaping* (Section 4.2 comes back to this subject).

In this paper, we present our incremental (shaping) method for learning in a multi-agent system and an experiment on which we tested our methodology. Section 2 discusses related work, Section 3 brings some background knowledge, Section 4 describes the approach we follow, and Section 5 describes the experiments conducted to test the viability of our approach and the results obtained. A discussion of our work follows in Section 6, highlighted by some comparisons with other similar works. Future directions and conclusive remarks end the paper in Section 7.

2. Related Work

2.1. On explicit coordination

Boutilier [7] has studied the problem of coordination from a theoretical point of view, using Multi-agent Markov Decision Processes (MMDP). In his framework, each agent can compute a globally optimal policy where only coordination problems are to be solved, i.e. when the optimal joint action is not unique. As he points out, such a coordination can be reached using social laws, communication or a learning mechanism.

A major drawback of this planning framework is that it requires not only a global perception, but also the ability to recognise other agents, which is rarely the case in a reactive MAS.

Modelling/Recognising Other Agents — Even without Boutilier's strong hypotheses, the ability to recognise other agents can be useful to understand their behaviours (learn their policies or determine their goals). Taking this information into account to choose an action may help improving the coordination and global cooperation. Yet, the other agent's model must be accurate since, as shown by Hu and Wellman [21], a bad model may be worse than no model. Historically, the modelling of other agents has mainly been used in competitive situations, other agents being viewed as opponents [12, 49]. Works often consider game theory and the simple prisoners dilemma [36].

Communication — Communication could also be used to solve the problem of explicit coordination. But attention must be paid to the fact that communication in itself has a cost [53]: it is a new resource to manage. Moreover, communication may have an important impact on the complexity of decision-making [33]. An important aspect of communication is the question of its content: intended actions, utility of actions, belief state, perceptions... ? A possible direction for tackling this question is that of learning (or "agreeing on") the interpretation of messages that have no prior common meaning [23, 19].

2.2. Reward Definition

Individual Viewpoints — It is very important to remember that defining the reward function is a crucial but difficult phase. A first point is that a "truly" autonomous agent should not rely on external help to receive reinforcement signals, but on an internal mapping from observations and actions to a scalar reward.[2] This is similar to the viewpoint adopted by Fernández and Parker [16], where the team tries to maximise the accumulated reward of all teammates despite the lack of communication. On the contrary, much research on multi-agent frameworks relies on complete observability and, as such, a common global reward function may be used by each agent, ensuring that they work toward a common objective.

Multi-Agent Credit Assignment Problem — In all cases, as we mentioned in Section 3.3, the multi-agent credit assignment problem (how to define individual reward functions when considering a global—cooperative—task) remains difficult, as can be observed in Tumer and Wolpert's work

on this topic (COllective INtelligence [52]). The main results obtained only concern situations where agents can be organised in sub-groups working on totally independent problems.

Modifying the Reward Function — The reward function can be modified to efficiently guide the learning agent. This has been applied in Matarić's work with multi-robots systems [27], where she takes the option of adapting the reward function and the agents' actions and perceptions to use a simple reinforcement algorithm. The reward is not a binary process (no reward / big reward at goal) but uses a "progress estimator", i.e. a smoother reward adapted to give very frequent hints to each individual agent. This method requires an important work from the human designer and is strongly task-dependent. Another example is that of Stone and Veloso's work on simulated robotic soccer [44], where they also choose to define a reward function giving an estimate of the team's progress.

The definition of a reward function—and how to use it—is an important problem that should always be kept in mind in multi-agent Reinforcement Learning. Moreover, theoretical advances are still required in the multi-agent framework, so as to ensure that individual reinforcement signals appropriately lead to an intended global goal.

2.3. Learning Algorithm

Partial Observability – In a partially observable setting, well informed decisions can be made using past observations to disambiguate the current situation.

One approach is that of computing a probability distribution over states (what gives a "belief state") [13]. But this requires knowing the underlying model of the system and can be computationally very expensive. Furthermore, in our setting, the underlying MDP changes when the complexity of the problem increases. This would make it very difficult to reuse any knowledge. Another approach consists in defining a policy on a sequence of past observations [15, 28], which is as scalable as the "instant" policy we are using. The main difficulty is that learning consumes much more time and memory when considering past observations.

To our knowledge, most practical experiments in multi-agent settings have only involved policies depending only on the immediate observation [27, 44, 16]. In this case, direct policy search (looking for stochastic policies) has had more success than dynamic programming (producing a deterministic policy) when both types of algorithm have been compared, as done by Salustowicz et al. [37] or as we experienced ourselves using Q-learning in earlier experiments. Work by Peshkin et al. [30] and Baxter

et al. [3] also make use of a direct policy search in multi-agent settings. A particular case of RL based on current observation is that of TPOT-RL [44], which seems specific to problems like the robot soccer it has been designed for. Indeed, it considers that the next state is not accessible when the ball is being passed to another player for example. As a consequence, a classical update using $V(s')$ to compute $Q(s,a)$—if the transition is $(s,a) \rightarrow s'$—is not possible, hence the idea of using a Monte-Carlo sampling to evaluate the expected future reward.

Multi-Agent Aspect – In a multi-agent setting, better results logically require the agent to reason about other agents' reasoning [21]. But this requires knowledge about this other agent's behaviour, while we have seen in Sec. 2.1 that it is not convenient to distinguish agents and that it would again lead to computationally expensive algorithms.

Architecture – In some challenging applications of Reinforcement Learning, a monolithic architecture seems often insufficient to tackle the complexity of the environment. A good way to go can be to decompose the problem according to different levels of abstraction (from low-level controllers to high-level strategies). This is quite common in robotics, with examples in foraging [27], robotic soccer [42, 1]. In this domain, different questions arise as: 'What should the various levels be?' and 'Can they be automatically defined [20]?' We do not make use of a hierarchical architecture, so as to focus on our proposed shaping methodology.

2.4. Automated Shaping

Existing work in the field of shaping [35, 34, 29] shows how efficient shaping can be, and gives some theoretical results on how a problem can be modified without altering the solution. Yet, how to modify a problem so that it is easier to solve remains an open question. It is probable that if one knows how to modify the problem, one probably knows a lot about the solution. Two experiments using reward shaping in a multi-agent application have been mentioned in Sec. 2.2 [27, 44].

Asada's completely observable model [1] made it possible to automate the definition of the agent's training. It was possible to evaluate the distance to the goal thanks to an ordering of perceptions. In the case of factored perceptions (a perception being described by a vector of variables), an interesting direction could be to analyse the frequency of changes in the variables, as done by Hengst [20] to discover hierarchy in Markov Decision Processes.

3. Background

In this section, we first describe the precise problem of Multi-Agent System design we want to address. We then present Reinforcement Learning, focusing on the details which are of interest to our work. Lastly, we explain which particular difficulties are met due to the use of RL in a MAS. This will lead to the original methodology proposed in Section 4.

3.1. Designing Multi-Agent Systems

We are interested in automating the design of cooperative Multi-Agent Systems by having each individual agent learn its own behaviour. As mentioned earlier, we have decided to work with very simple reactive agents for complexity reasons. Besides, it allows us to concentrate on the learning aspect of the design.

Among many possible choices, our agents can be characterised as:

- **reactive**: Agents have "reflex" behaviours, act based on current observation only (memoryless behaviour).
- **situated with local perception**: Even if "partial" observations are a disadvantage when learning, their "local" aspect is of benefit as it limits the risk of combinatorial explosion (only a few elements are perceived).
- **possibly heterogeneous**: Although they may have the same abilities for perception and action, each agent can acquire a different behaviour from the others, as agents learn individually in the adopted learning process.
- **cooperative**: All agents share the same goal and they *will* have to coordinate to reach it.

Agents may have other characteristics (such as communication skills), yet we do not have any preferences on them, as they do not play any crucial role in our approach.

The question raised is then the following: given a global task (i.e. a task that a priori requires a global view of the system in order to be solved), how to design –independently and in decentralised way– the individual behaviours of agents having a local and partial view of their environment ? For the reasons mentioned in the introduction (formalism, top-down approach with no supervision), we propose to tackle this problem through Reinforcement Learning (interested readers

can find an introduction to Reinforcement Learning written by Sutton and Barto [45]).

3.2. Reinforcement Learning

3.2.1. *Markov Decision Processes*

We first consider the *ideal* theoretic framework for Reinforcement Learning, that is to say Markov Decision Processes [45, 32].

Let $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r \rangle$ be a Markov Decision Process (MDP) where $\mathcal{S}$ is a finite set of **states** and $\mathcal{A}$ a finite set of **actions**. This process is Markov since state transitions are ruled by the **transition function** $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ with $\mathcal{T}(s, a, s') = Pr(S_{t+1} = s' | A_t = a, S_t = s)$ (Figure 1 presents MDPs as a graphical model). r is a mapping from $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$ that defines the **reward** gained by the system after each state transition. An action **policy** π is a mapping from states to distributions over actions ($\pi : \mathcal{S} \to \Pi(\mathcal{A})$, with $\Pi(\mathcal{A})$ a probability distribution over actions). The problem is then to find a policy optimising a performance measure based on the reward, named **utility** and denoted V. Typically, the utility can be the sum of rewards to a finite horizon, the weighted sum of this reward on an infinite horizon ($\sum_{t=0}^{\infty} \gamma^t r_t$ where $\gamma \in [0, 1)$), or the average reward gained during a transition.

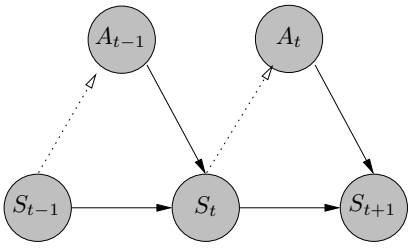


Figure 1. Graphical Model of an MDP. Dotted arrows indicate that an action is chosen depending on a state. Everything here is observable (indicated by the grey background), but rewards are not represented (to keep the figure simple).

In this paper, only *model free* RL [45, 24] is considered: agents look for optimal policies with no knowledge of the system's model ($\mathcal{T}$ and r). If a model could be computed (when the dynamics of the system are known), multi-agent problems usually lead to huge state spaces (size exponential in the number of agents). The general principle of model-free RL algorithms is to have the learning agent evolve in its environment and gather results of experiments (in state s_t, action a_t may lead to state s_{t+1} with reward r_t). Through stochastic approximation methods, the agent can then directly learn (using Q-learning [50] or Sarsa [45] for example) a value function on each of its states and deduce an optimal deterministic policy (whereas general policies may be stochastic), the

Markov property being a guarantee that there is no local optimum in this process and that one deterministic optimal policy exists.

3.2.2. *Partially Observable MDPs*
In our framework, agents only deal with partial observations. Therefore, they do not have access to a complete state of the system and are rather facing a Partially Observable Markov Decision Process (POMDP). Such a POMDP is defined by adding to the $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r \rangle$ tuple a finite set Ω of possible observations, and an observation function $\mathcal{O}$ linking states to the observations they may cause: $\mathcal{O}(s, o) = Pr(O_t = o | S_t = s)$ (see Figure 2). Besides, in our model-free problem, an agent has no knowledge of the underlying MDP ($\mathcal{S}$, $\mathcal{T}$ and therefore $\mathcal{O}$).

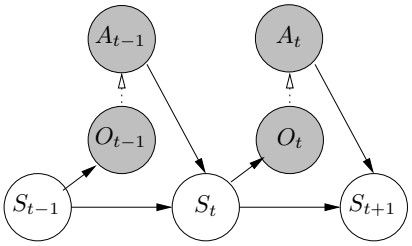


Figure 2. Graphical Model of a POMDP (see also Figure 1). In the case of a POMDP, states are hidden (represented by a white background). Note: the choice of action depends on the current observation only.

What an agent experiences is then only linked to observations instead of states. As a result, the process (O_t) obtained is not necessarily Markov anymore ($Pr(O_t|O_{t-1}) \neq Pr(O_t|O_{t-1}, O_{t-2}, \ldots)$), and there may not be a deterministic policy among the optimal solutions. This requires looking for stochastic policies of the form: $\pi : \Omega \longrightarrow \Pi(\mathcal{A})$. The dynamic programming algorithms we mentioned for MDPs are not suitable in this new framework (since the Markov property is not satisfied), but may be replaced by policy search algorithms as –for example– the online policy-gradient we have used (based on Baxter et al. [3, 4]). Early experiments have also been performed with Q-learning, showing that looking for a stochastic policy leads to better results, and turning a deterministic policy in a stochastic one using a soft-max is no satisfying solution.

This choice of a policy-gradient is not so usual, as it often appears to be sufficient to use dynamic programming algorithms, which are an easy pick in many cases. Yet policy-gradients are not only more appropriate in theory (optimising in the space of stochastic policies), but appear in recent works as very promising in problems involving function approximation techniques [46, 31] and in probabilistic planning [9]. In both cases, policy-gradients make it possible to optimise a policy in a large

state-action space by using either function approximation methods or a carefully chosen controller.

Learning is all the more difficult under these new conditions, as there may now be local optima into which an agent can permanently fall. Interested readers may also refer to papers by Littman et al. [26] and Singh et al. [39] to learn about the use of MDP algorithms in partially observable settings, and by Jaakkola et al. [22] for another algorithm that could be used instead of the mentioned online policy-gradient.

3.3. RL and MAS

As pointed out by Boutilier [7], the evolution of a Multi-Agent System can be modelled as an MDP if considered from an external point of view: depending on the *global* system state, there is a *joint action* to choose (the set of all agents' individual actions). Indeed, most theoretical works on Reinforcement Learning in MAS are grounded on this formalism, as can be observed from thorough reviews by Stone and Veloso [43] and Shoham et al. [38] (the work by Gmytrasiewicz and Doshi [18] gives a counter-example).

Yet, such an approach appears not to be reasonable for practical applications. The reason for this is twofold:

- The global state space will grow exponentially and will shortly be unmanageable (this is already often the case in simpler mono-agent settings).
- Realistic perceptions are generally limited to a local view of an agent's environment.

Our motivation to deal with these two limitations is the main argument for working in a decentralised and partially observable framework.

Unfortunately, as shown by Bernstein et al. [5], solving the problem in a non-centralised way when the agents only have a partial perception of the system's state is NEXP-complete, i.e. there is provably no polynomial algorithm to solve the problem. The additional complexity of our problem –compared to the classical MDP setting– is due to three major difficulties being faced:

1. **Partial observability.** As already mentioned, each agent's perception is local, which makes them uninformed of the global state of the problem. As such, the problem at hand belongs to the class of *partially observed* Markov decision models (see Section 3.2.2).
2. **Non-stationary transitions.** Reactive agents with a local view of the environment cannot use joint actions to solve the problem. In

fact, other agents are hardly predictable elements of the environment: as agents learn their policies simultaneously (in our work), each agent lives in an environment with non-stationary transitions (other agents are part of the agent's environment).

3. **Multi-Agent Credit Assignment**[3]**.** In a Multi-Agent problem, an important difficulty is to reward agents appropriately. Intuitively, an agent whose decisions have not contributed to a recent success of the group should not get any reinforcement signal. Yet, if an MDP point of view is adopted -all agents getting identical rewards- an agent may "understand" which decisions are really useful as only some of them regularly lead to successes.

   In our framework, reinforcement signals are internal to the agent, depending on its immediate observations. Thus, we face the problem of defining individual reward functions that correctly represent a common group objective (a question raised in more details in the COllective INtelligence approach [51, 47]).

   Note that in practice, e.g. in traffic control, local rewards make learning often easier, even if they may not lead to an optimal controller.

Classical stationary Partially Observed Markov Decision Processes are nearly impossible to solve when there are more than a hundred states [15]. The combination of the first two problems (i.e non-stationarity and partial observations) makes the problem non-solvable without using approximations. The multi-agent credit assignment issue is a rather independent question compared to non-stationarity and partial observations. We simply assume in this paper that the reward functions employed appropriately leads to a cooperative system.

4. Shaping: incremental reinforcement learning

As there are no exact algorithms looking for optimal behaviours for the class of agents we consider, we will use approximation methods. Even if the learning conditions are worse than those of a POMDP (see Section 3.3), we have decided to make use of an algorithm suited to this partially observable mono-agent situation. Each agent will use a gradient descent RL algorithm (or policy-gradient) [3, 4] suitable for on-line learning in a model-free POMDP, though it is not designed for use in our multi-agent framework.

After a brief description of this policy-gradient algorithm, we will focus on the main point of this paper: the use of a shaping method

to design Multi-Agent Systems. This presentation is done by first introducing the general idea behind shaping and then detailing how this approach is here adapted to a MAS.

4.1. Local learning algorithm used

As stated earlier, *each* agent uses its own policy-gradient algorithm to learn its policy (Q-learning was tried in early experiments, but was found to be unstable, most likely as it was looking for deterministic policies in a non-Markovian framework). We use an on-line version of the algorithm (presented here in its usual *mono*-agent setting).

As proposed by Baxter et al. [3, 4], a policy π depends on a set of parameters Θ. This leads to an expected utility $V(\pi_\Theta) = V(\Theta)$. The policy-gradient leads to obtaining a locally optimal policy by finding Θ^* that makes the gradient equal to zero: $\nabla V(\Theta^*) = 0$.

The set of parameters we choose is $\Theta = \{\theta(o,a), o \in \Omega, a \in \mathcal{A}\}$, with $\theta(o,a)$ a real valued parameter. The policy is defined by the probabilities of taking action a in state o as:

$$\pi_\Theta(o,a) = \frac{e^{\theta(o,a)}}{\sum_{b \in \mathcal{A}} e^{\theta(o,b)}}$$

Although this algorithm is theoretically not suited to MAS, cooperating agents prove to simultaneously evolve toward coordinated behaviours. Some readers may also be interested in the fact that such an algorithm may be used to learn controllers for several cooperating agents in a centralised way: to that end, each agent A_j has to be described by a subset Θ_j of parameters linking its observations to its actions [2, 30, 8].

4.2. Incremental RL: Shaping

To speed up learning and improve the efficiency of resulting behaviours (mainly by avoiding being stuck in low local optima), we propose applying a progressive learning method. This approach, inspired by works in psychology [40, 41] (Randløv and Alstrøm [35] give an historical introduction), has been employed successfully in mono-agent RL, but only in completely observable MDPs [29, 34].

The main idea is to ease the learning task by turning the problem the agent is facing into a simpler problem, and then progressively coming back to the original one. In fact, as far as Markov Decision Processes are concerned, a problem may be modified in various ways, as:

– The reward function may be redefined [35, 29, 25]. This is even done intuitively in some cases by rewarding the agent when it goes

toward its goal while a reward when the goal is reached should be sufficient. Note that this may be hazardous (agent going back and forth to accumulate reward), but there are some theoretical guarantees [29].

– The physics of the system may be altered [34]. This corresponds to modifying the MDP's transition function. A simple example is that of adding training wheels to a bike, and then progressively raising them [34].

Whereas very appealing, the idea of shaping is not so easily applied: it may be difficult to determine what is a "simpler" problem. As noticed in Laud's PhD thesis [25], shaping is generally a way to bring prior knowledge to a learning agent. In a way, it often turns Reinforcement Learning into a slightly supervised learning.

In a third shaping approach (although its author does not mention shaping) an algorithm can successfully find "simpler" problems by itself. In this work by Asada [1], the learning agent is helped by being first confronted by states "close" to the goal to achieve, and then being progressively put in more and more complex situations (far from this goal). The result is to help the spreading of rewards in the value function (or Q-values). In this particular approach, and if a model is known (transition and reward functions $\mathcal{T}$ and r), the definition of the training used with the agent may be automated, since states close to rewarded transitions can be identified. This makes it effectively possible to determine what is a "simpler" problem.

This third type of shaping is not completely orthogonal to reward-based shaping. It relies indeed on a progress estimator (since it uses start states close to the goal), and using a progress estimator is and ideal way to shape a reward function. Yet they can lead to different behaviours: reward-based shaping won't be efficient in a maze, since the distance to the goal is often a bad progress estimator, while Asada's shaping acts more as a backward search as it starts from the goal. Transition-based shaping is often difficult to implement and to benefit from: in the mountain-car problem [45], starting from a flat mountain and progressively raising it does not help, as there is a critical point where the policy has to switch from moving directly to the goal to moving back and forth to get momentum.

This comparison between these three shaping approaches led us to prefer Asada's method. But different contexts would lead to different choices. A second, and stronger, argument is that it naturally extends to our shaping approach for Multi-agent Systems as described hereafter.

Note: the term "shaping" sometimes appears to refer to design methods mixing engineering and automatic design algorithms (as reinforcement

learning and evolutionary algorithms) [14]. This is slightly different from the terminology used in our context.

4.3. Shaping for Multi-Agent Systems

To speed up learning and reduce the problems of complexity and credit assignment[4], we propose a methodology for shaping in a multi-agent setting. A first step follows the same idea as in Asada's work, and a second one is based on a growing MAS.

Growing Task Complexity
Incremental learning is possible along the complexity dimension of the problem. Agents are pitted against harder and harder tasks. First tasks are "near" (in term of number of actions) positive reinforcement positions, then further and further away. While the learning progresses, agents have more and more freedom of action and can explore their environment further ahead.

To be more precise, a *step* consists in all agents making one move simultaneously. Then, we define a *trial* as a sequence of n steps beginning in a given situation (close to the goal at the beginning). This *trial* must be repeated sufficiently (N times) to be useful. This succession of *trials* will be called an *experiment* for our agents. The trainer has to define a sequence of progressive *experiments* to help learning. Algorithm 1 gives a more formal presentation of the process, which will also be usefully illustrated in our experiments (Section 5.2).

Growing MAS
This phase is more specific to the conception of Multi-Agent Systems. It consists of learning with a reduced number of agents, and then multiplying them. This is simply done by cloning the first agents obtained, and then letting them all learn again. But one should not forget that all agents always learn their own behaviours, clones may therefore evolve in different ways.

Simultaneously, we also add other objects in the environment, which –from an agent's point of view– will increase the complexity of learning in a similar fashion as adding other agents. Indeed, if the goal remains the same, more objects are interacting while the agent only perceives few of them. An important remark at this stage is that the agents' internal architecture must be adapted to a variable number of objects/agents in the environment. This problem has been answered in our experimentation with a rather simple method based on the notion of local perceptions (see the problem description in Section 5.1 for more details). This shows once again how important locality is in Multi-Agent Systems, even if this restricts each agent's knowledge about the world.

Algorithm 1 Complexity shaping

Require: A system Σ consisting of a set $\mathcal{A}$ of agents and an environment.
A training T.
1: **for all** experiment E of the training T **do**
2: $e =$ trial defined in E by:
3: $\star$ s_0 initial state of the system
4: $\star$ n number of steps to accomplish from s_0
5: $N =$ number of times e has to be repeated
6: **for all** trial $i \in [1..N]$ **do**
7: Put system in state s_0
8: **for all** step $j \in [1..n]$ **do**
9: **for all** agent $\alpha \in \mathcal{A}$ **do**
10: Give α its perceptions o_t^α
11: Let α learn from its last experience
12: Ask α for its decision a_t^α
13: **end for**
14: Simulate system's evolution: $\Sigma_t \rightarrow \Sigma_{t+1}$
15: $t \leftarrow t + 1$
16: **end for**
17: **end for**
18: **end for**
19: **repeat**
20: [lines 9 to 15]
21: **until** stopping criterion of the learning algorithm verified
Ensure: Policies of agents in $\mathcal{A}$.

Note: the question of designing such a *scalable* agent (able to handle a variable number of perceptions and motivations) also led to other developments [10] that fall out of the scope of the present paper (see discussion in Section 6).

5. Experimenting with shaping

An application of the shaping method presented in the previous section is given in the experiments described here. After a short description of the problem, we give the details of the experiments conducted, which separately analyse both aspects of the shaping used (growing complexity and growing MAS), and also study the influence of a MAS of variable size.

5.1. Problem description

The task chosen involves agents (either yellow or blue) in a grid world whose goal is to push light green cubes against dark green ones[5]. When two agents coordinate their movements to attain this goal –pushing *together* a pair of cubes– both cubes temporarily disappear to randomly reappear elsewhere on the grid. Simultaneously, agents responsible for this fusion receive a positive reward. The agent's objective is then to merge pairs of cubes as often as possible, not just foraging.

Agents' description

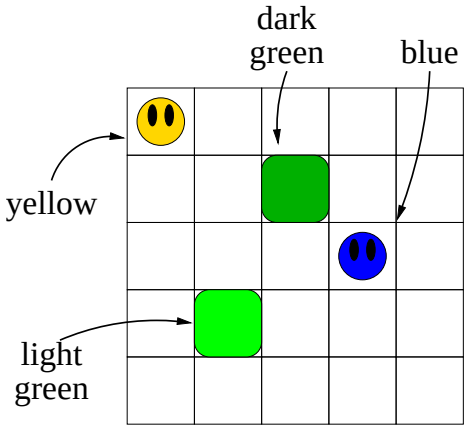
- **actions**: Agents only have four possible actions corresponding to moving North, East, South and West (they always try to move). Agents can push other agents and other cubes, which makes the consequences of their actions stochastic. Indeed, when several agents are influencing a cube's movement (for example), only one –randomly chosen– influence will have a real effect.

- **perceptions**: As shown in Figure 3, an agent's perceptions are made up of the following information:

  - `dir(oa)`: direction of nearest agent of the opposite colour `(N-E-S-W)`,
  - `dir(cl)/dir(cd)`: direction of nearest light cube (and dark cube) `(N-NE-E-SE-S-SW-W-NW)`,
  - `near(cl)/near(cd)`: is there a light cube (dark cube) in one of the eight nearest positions `(true|false)`.

  Combining these, there exists a maximum of 1024 observations (some combinations of perceptions are not possible). We reduce this number to 256 by using symmetries of the problem. This number is small compared to the 15 249 024 states of the 8×8 totally observed centralised problem, and also it is *independent of the world's size*.

- **reward**: The reward function we have chosen eases the credit-assignment problem. When an agent takes part in the fusion of two cubes, it gets a reward (+5), while the reward is zero the rest of the time. Indeed, an agent knows when it has taken part in such a fusion as this corresponds to the event of seeing a pair of aligned blocks disappearing when the agent moves in their direction. A "global" reward would not be very consistent with agents having only local information.
  There is no guarantee that this local reward function leads to a



agent	dir(cl)	dir(cd)	dir(oa)	near(cl)	near(cd)
yellow	S	E	SE	no	no
blue	W	NW	NW	no	yes

cl: light cube - cd: dark cube - oa: other agent

Figure 3. perceptions' examples (two agents in a simple world)

complete cooperation, as would be the case with a global one. Only experiments can easily confirm/infirm the choice we made.

We conclude by some remarks about the simulation itself. To focus on the learning problem, we have implemented simple mechanisms to avoid some unrelated problems. For example, cubes cannot go on border cells of the grid, neither by being pushed, nor when reappearing on the grid (this could otherwise lead to blockings, with cubes that could not be moved away from the border).

5.2. The 2-agent and 2-cube case

5.2.1. *Reduced problem*

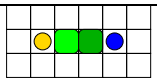
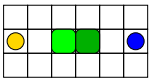
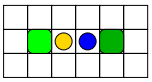
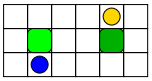
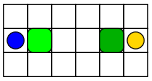
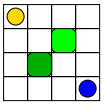
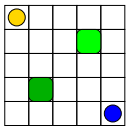
The first step in our incremental learning scheme is to use a small sized world with a minimal number of agents and objects: one agent and one cube of each colour. Then, beginning with positive reinforcement situations, the agents will face harder and harder problems.

5.2.2. *Progressive task*

For our problem, there is only one situation that leads to a positive reward. This situation is thus the starting point in the shaping process Agents face a sequence of tasks to learn before ending in a standard 8×8 environment where they keep on learning.

Table I shows a sequence of *experiments* we used to help our agents in their *training*. The *starting configuration* of the first *trial*, on a 6×3 world, needs only one move to reach the goal, each agent pushing toward the cubes. However, they have up to 6 *steps* (the duration of a *trial*) to reach it (so they can explore different moves), and they can try this 150 times (duration of this *experiment*).

Table I. The sequence of experiments we used for incremental learning.

Starting configuration	n (moves)	N (trials)
	6	150
	6	100
	10	150
	20	150
	20	150
	100	15
	100	15

In order to estimate the efficiency of our approach, a standard learning (from scratch) is compared to the incremental learning (shaping) proposed. By "from scratch", we mean the use of the same policy-gradient algorithm, but always starting with a blank policy i.e., with no prior knowledge. In both cases, we count the number of cube merges accomplished during 1000 time steps to evaluate the quality of the agents' policies, these agents being put in an 8×8 environment once the training has ended (when there is one). We will now observe and analyse the results obtained.

Results
Figure 4 shows the evolutions in the pair of agents' efficiency with and without using shaping (average of 10 independent simulations). The curve representing the efficiency of learning after a period of assisted training only begins after the 12000 time steps of this training (Table I).

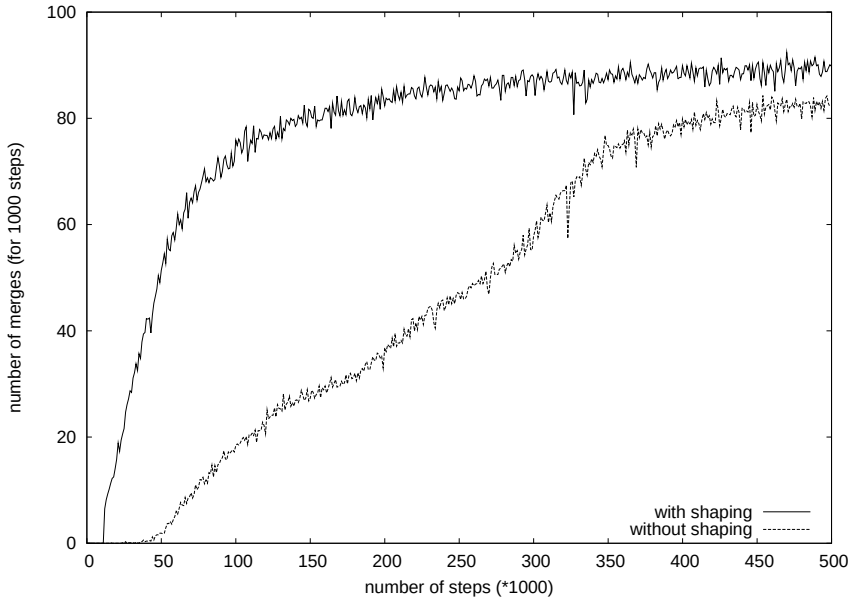


Figure 4. 2 agents, 2 cubes: learning from scratch vs. shaping

In both cases, agents tend toward behaviours allowing about 90 merges each 1000 time steps (on average, two cubes are merged every 11 time steps). On the other hand, once the training time has elapsed, the convergence toward very good performance is achieved notably faster than through "standard" learning. If agents do not learn everything through the script that guides them during early times, at least do they benefit from it to efficiently learn their policies.

Designing the Sequence of Experiments
The sequence of experiments should provide problems of growing complexity. Here, it is rather easy to design appropriate starting situations, since it suffices to move agents away from the blocks and blocks away from each other. Because there is a positive reward only in one state, an increasing distance to this state (in the state space) corresponds generally to an increasing complexity. We further discuss the problem of designing this sequence in Sec. 6.5.

5.3. More agents and more cubes

We now consider the use of more agents and more cubes. As done previously, learning with shaping will be compared with standard learning (without shaping). But we will begin by considering the efficiency of the policies learned in the previous experiments (with 2 agents and 2 cubes, which will be denoted $2a2c$) when they are reused in more populous worlds.

Note: Here, agents that do not start learning from scratch (either to simply reuse policies or to improve them) have their policies initialised with policies learned in $2a2c$.

5.3.1. *Results: simple reuse of policies*

In this part of our work, the first interest was to check the efficiency of different numbers of agents having to deal with different numbers of cubes (taking the same number of agents (or cubes) of each colour). So, we first use agents whose fixed behaviours have been learned in the $2a2c$ case (as illustrated on Figure 5), i.e. which are not supposed to be very efficient with more cubes. Nevertheless, this gives them enough good reactions to have some good results.

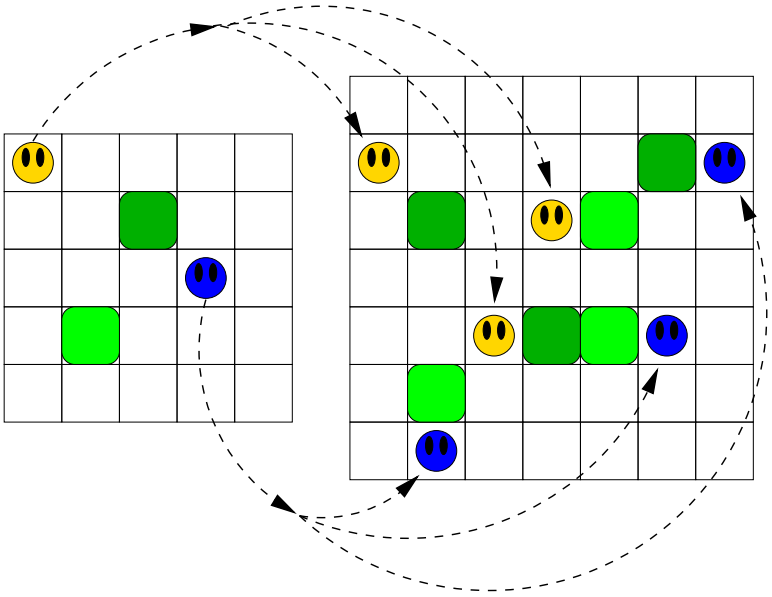


Figure 5. Replicating behaviours from 2 to n agents.
Policies are simply copied from the two original agents into their "clones".

Several tests were carried out with 2, 4, 6, 8 or 10 cubes and 2, 4, 8, 16 or 20 agents (always in a world of size 10×10). We used series of 1000 consecutive time steps, these series beginning in random configurations.

And as blocking situations could sometimes occur, 100 such series were made in each case in order to compute an average efficiency.

Table II-a gives the average efficiency in each of the 25 situations (the efficiency is the number of merges made in 1000 steps). These results can be compared with those from Table II-b that measure the efficiency of agents having a completely random behaviour but placed in the same environments. It then clearly appears that behaviours learned by agents having to handle 2 cubes are still efficient in more complex environments. Through these results, a trend seems to take shape: except for 2 cubes, a growing number of agents improves the results up to the point where agents cramp each other and bring too many coordination problems.

Table II. Agents' mean efficiencies (number of merges in 1000 steps)

a. Reuse of $2a2c$ agents

cubes	agents				
↓	2	4	8	16	20
2	40.4	30.0	20.0	12.7	11.0
4	7.6	17.1	17.5	13.9	12.9
6	3.4	11.2	14.7	15.7	16.5
8	1.9	8.6	13.5	15.9	18.0
10	1.6	6.7	11.0	17.7	20.6

b. Random agents

cubes	agents				
↓	2	4	8	16	20
2	0	0	0.1	0.3	0.4
4	0	0.1	0.2	1.2	1.8
6	0	0	0.5	1.9	3.6
8	0.1	0.2	1.0	4.1	6.0
10	0.1	0.3	1.1	6.1	7.3

For a given number of agents, and with a growing number of cubes, there seem to be a threshold beyond which additional cubes lead to a deterioration of the group's performance. A more qualitative analysis of observed behaviours has shown that agents remain stuck in sequences of oscillatory movements. Because of their partial perceptions and of their lack of communications, agents seem to have difficulties working together on a same subset of cubes and constantly hesitate between two options that alternatively appear to be of interest.

5.3.2. *Results: Incremental learning along the number of agents*
Rather than just reusing behaviours learned for a 2 agents and 2 cubes situation in more cluttered environments, the second phase of our methodology –as defined in Section 4.3– consists in using such behaviours as a starting point to adapt agents to new situations by continuing the reinforcement learning.

To assess this process, we compare the learning curve when the agents are initially novices (starting from scratch) to that when the agents are endowed with $2a2c$ policies. Figures 6 a, b, c and d show the results in four distinct situations.

The learning for novice agents is difficult, especially in situations a and b because the size of their environment (larger than the environment used in Section 5.2) leads to scarcer non-zero rewards. In fact, in cases a and b, novice agents do not learn anything and their behaviours are the same as randomly moving agents. Indeed, in each case, novice agents evaluated at time $t = 0$ are simply random agents having spent 1 000 time steps in the environment. With more agents, even novice agents can learn, as the frequency of rewarded events is higher. This is an interesting phenomenon since, otherwise, more agents makes learning harder. It can be related to difficulties encountered when learning to solve planning problems: a reward being provided only when a goal is reached, a bigger state space leads to infrequent rewards and therefore a harder learning task.

Coming back to the initial problem in this study, these experiments clearly confirm that agents having some experience from a problem of the same kind have a much faster progression than beginners. In case a, the efficiency level is even optimal from the beginning (which is not surprising as the conditions are nearly the same as in the initial learning problem). In addition to an improved learning rate, the maximum efficiency reached by the agents through shaping is by far better than performances obtained when learning from scratch. The reason for this is that agents in a noisy environment (noise here due to a large number of objects/agents) find it difficult to learn that they should sometime go on the other side of a pair of cubes, whereas this knowledge is present in $2a2c$ behaviours and can then be reused.

6. Discussion

6.1. On explicit coordination

Our experiments show that this work could benefit from explicitly addressing coordination problems. In particular, it is often the case that two agents, when placed in a world with more than two cubes, are not able to coordinate so as to push the same pair of cubes. Moreover, when too many agents are present, they easily work at cross-purposes. Making appropriate decisions would require taking into account many more agents and objects than each agent can through their limited perceptions. Instead, agents act often in an inconsistent manner (nearly randomly) because they are not "paying attention" to most aspects of their current situation but act greedily.

Other Agents' Modelling and Communication — Considering both other agents' modelling and communication, it is important to notice

how computationally expensive these approaches can be, as they often lead to huge growths of state-action spaces: in one case agents try to guess what others will be doing, and in the other case agents have to decide when and what to communicate (new actions) and have to incorporate received communications in augmented perceptions. Moreover, the work presented in this paper focused on reactive agents, whereas these approaches are quite "high-level" (imply more cognitive agents).

6.2. Reward Definition

Reward Shaping — Modifying the reward function to help learning—as described in Section 2.2 and as encountered in Matarić's work [27]—is known as *reward shaping*. As a shaping process, it shares some similarities with the "progressive training" part of our approach, but Matarić's experiments can hardly be compared with ours as this real robotic problem involves rather high-level macro-actions: each macro-action is a complete behaviour, which would correspond to a sequence of actions in our case. Moreover, interactions between robots are not strong in the foraging task. There is no coordination problem to solve.

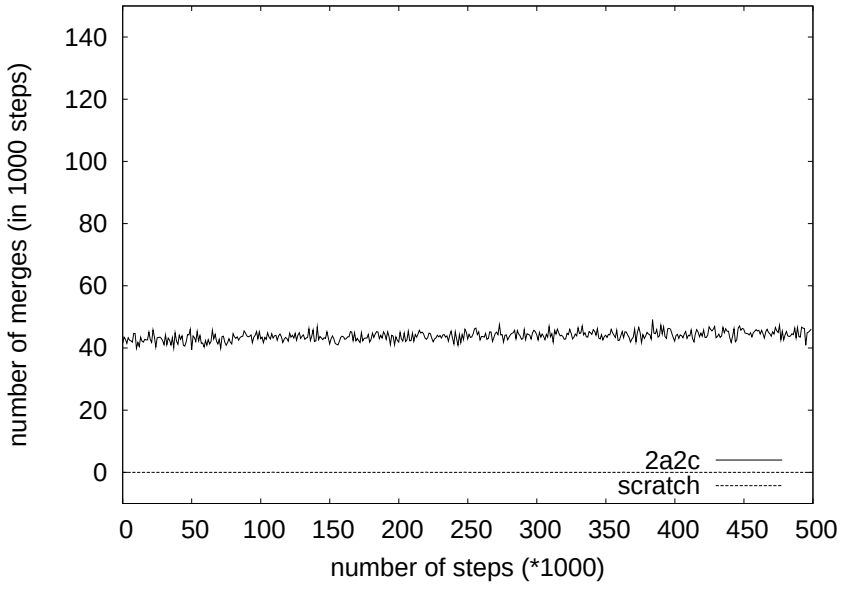
To come back to the other example of Stone and Veloso's work on simulated robotic soccer [44], it could be interesting to try our "cloning" approach on robotic soccer (starting with a 2×2 game for example), but a major difference that could make things more difficult is that the soccer-playing agents will probably end up having different roles, each being assigned a specific position.

6.3. Scalable Architecture and Multiple Motivations

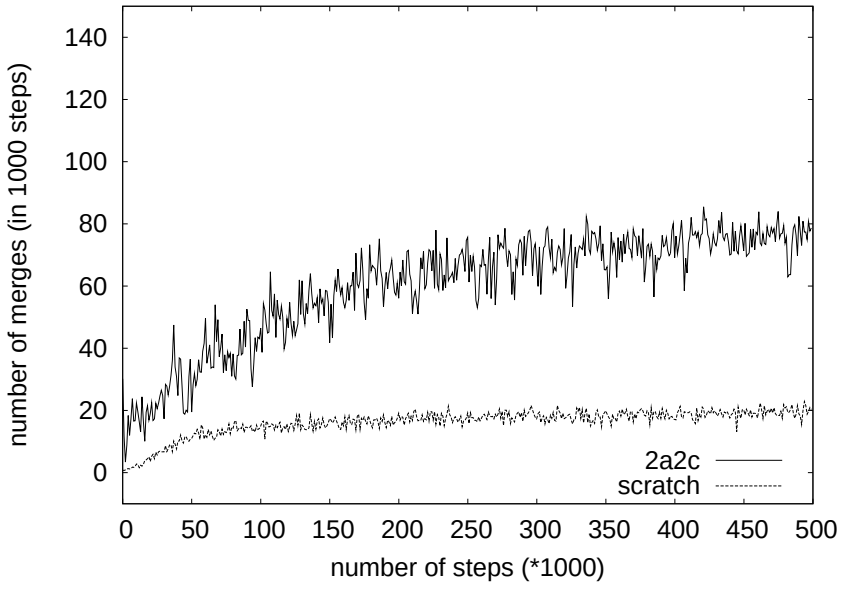
An important aspect of our method is that it requires agents whose internal architecture is adapted to a variable number of objects or agents in their environment.

In our experiments, this constraint is satisfied by designing a perception process always selecting the same number of visible objects (the closest ones). This prevents from taking into account more than two objects and one agent simultaneously, and makes it impossible to have a different behaviour depending on which agent is considered. Moreover, as the visible agent may be different from one time step to another, it does not seem appropriate to make decisions based on a sequence of observations.

We have tried to address this issue by working on decision making when multiple motivations are faced. Because the Multi-Agent Systems considered are reactive and with local perceptions, each agent is often facing multiple parallel motivations, and has thus to make a choice in a "multi-criteria" problem. In our block-merging example, limiting

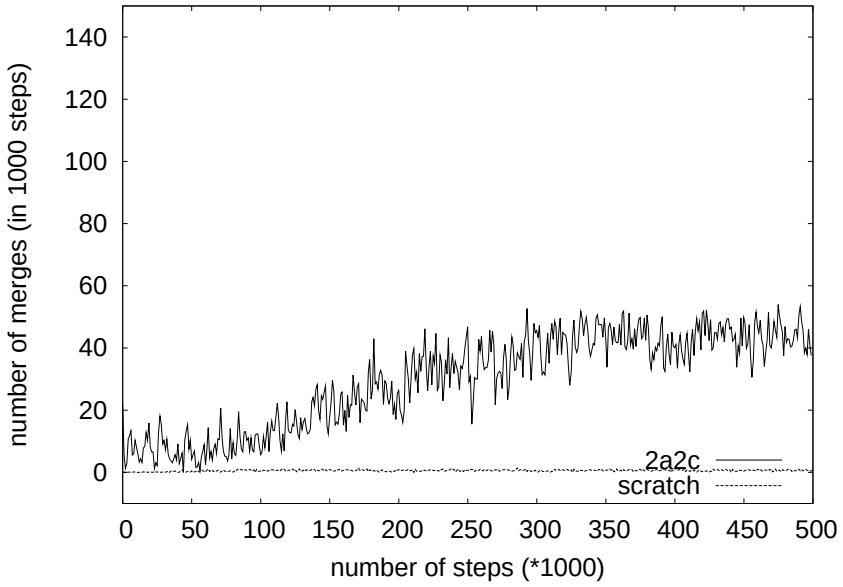


a- $2a2c$

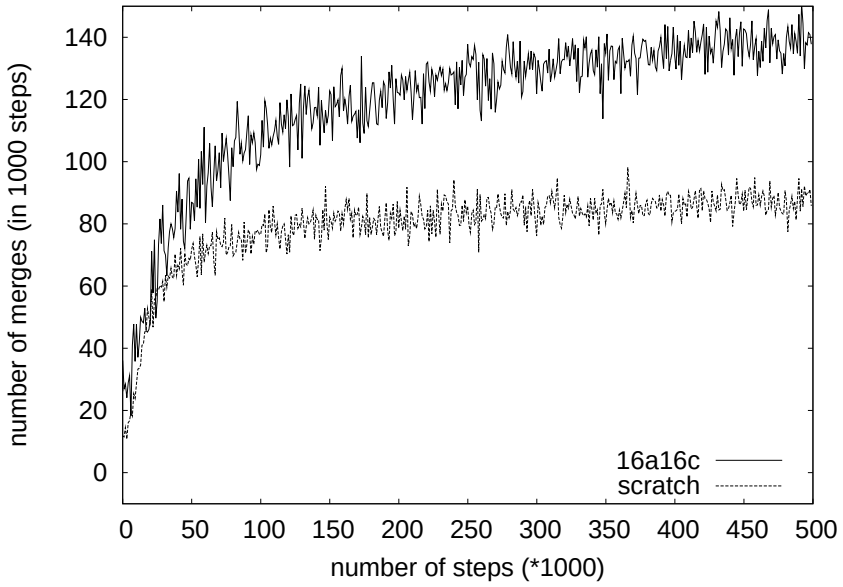


c- $8a8c$

Figure 6. Comparison between standard learning (from scratch) and incremental learning (reusing behaviours learned in a $2a2c$ problem in a small grid) in various environments. During the learning process, the efficiency is measured by the number of merges done in 1 000 time steps. Each curve is the average of 100 simulations.



b- $4a4c$



d- $16a16c$

Figure 6. (con't)

the perceptions ensures that only one motivation is faced. But work on "action selection" [48] could be a way to address this particular problem. We conducted some work in a mono-agent framework, obtaining an agent able to find which basic behaviours it should use [10, 11]. Applying our method in a multi-agent framework still needs to be done.

6.4. Learning Algorithm

For the reasons mentioned in Section 2.3, we have decided to use a reinforcement learning algorithm mapping the immediate observation to a probability distribution over actions. Some experiments with more classical algorithms (Q-learning) producing a deterministic policy have shown that this is not a viable solution in our toy problem: the ambiguity on current situation leads agents to looping behaviours (agents pushing their blocks and turning around them). Let us recall that both the partial observability and the multi-agent setting are reasons for deterministic policies to fail.

6.5. Automated Shaping

Coming back to the shaping process proposed in this paper, it appears to be mainly constrained by the need to decompose the problem at hand into less and less complex tasks (Section 4). The problem is partially made easier in our example as the starting points of this decomposition are rare positive reward situations (the reward being null most of the time). Yet, it remains difficult because of the partial observations. Moreover, our shaping method relies on cloning agents, which is obviously limited to settings with relatively homogeneous Multi-Agent Systems (with a few types of agents). This assumption seems reasonable in many domains.

Also, considering multi-agent systems, our approach involving cloning agents raises issues regarding task/role allocation: is it easy for agent to specialise for a particular role with this type of shaping ? If the cloning process is creating copies of the same policy, further learning can lead to a differentiation. But task/role allocation does not necessarily imply having agents with heterogeneous policies. If agents with identical policies have different perceptions or internal states, they can choose to take on different roles. But this view is quite different from most research on task allocation [17], where tasks are explicitly defined, appropriate behaviours are known for each task, and the problem is for the agents to decide who will perform which task.

7. Conclusion

In this paper, we have addressed the problem of automatically designing Multi-Agent Systems made of reactive and cooperating agents, each acting as an independent learner. We propose the use of Reinforcement Learning algorithms so that each agent individually adapts its local behaviour in order to achieve a global task. Up to now, this learning problem has no theoretical solution because of its decentralised aspect and because of the partial perceptions of the learning agents.

To get around this difficulty, we have emphasised the use of an incremental learning (shaping) scheme where more and more agents are faced with harder and harder problems. This method is quite generic as it can be adapted to any task without adapting the agents to the particular problem at hand. Its applicability is mainly limited by the agents' architecture, i.e., how they handle perceptions. It is indeed required to be able to re-use a learned policy in a new setting with more objects. This is feasible with reactive agents using simple sensors (as in our experiments). It could be extended to more complex action-selection architectures based on a combination of simple behaviours [10].

We have tested our approach on a simulated environment where agents have to coordinate in order to reach a shared goal: the fusion of different coloured cubes. The experiments support our framework as they show the efficiency of incremental learning. Incremental learning leads to better learning rates than unassisted learning from scratch. Furthermore, it is more efficient to learn a more complex task after an initial stage of incremental learning than learning directly this more complex task from scratch: this shaping helps overcoming local optima.

Still, there is room for improvement. We have discussed several ways to overcome these limitations, like using communication for addressing explicit coordination or modelling other agents to better adapt to their behaviours. Defining individual reward functions representing a common goal also remains a crucial issue. Yet, the main improvement regarding our shaping approach would be to automate the definition of training phases in our progressive learning (maybe by analysing perceptions as done in some recent works [20]).

Furthermore, this work was a good opportunity to use a direct (stochastic) policy search algorithm in cooperative agents with local perceptions. It also made it possible to observe some phenomena in such a multi-agent learning setting: the main one being that a large number of agents induces more interactions and thus a faster learning, but not necessarily with the best behaviour since the incremental process is able to produce better behaviours.

Notes

[1] Note that most other studies make the opposite assumption of a total perception by each agent.

[2] But an agent learning in "controlled conditions" could get its reward from a third party, which is not our choice.

[3] Not to be confused with the *temporal* (mono-agent) credit assignment problem: which past decision should be reinforced because of the present reward ?

[4] Mono-agent credit assignment problem mentioned in a previous footnote.

[5] From an agent's point of view, light and dark green cubes cubes are interchangeable.

References

1. Asada, M., S. Noda, S. Tawaratsumida, and K. Hosodaal: 1996, 'Purposive Behavior Acquisition for a Real Robot by Vision-based Reinforcement Learning'. *Machine Learning* **23**(2–3), 279–303.
2. Bartlett, P. and J. Baxter: 1999, 'Hebbian Synaptic Modifications in Spiking Neurons that Learn'. Technical report, Australian National University.
3. Baxter, J. and P. Bartlett: 2001, 'Infinite-Horizon Policy-Gradient Estimation'. *Journal of Artificial Intelligence Research* **15**, 319–350.
4. Baxter, J., P. Bartlett, and L. Weaver: 2001, 'Experiments with Infinite-Horizon, Policy-Gradient Estimation'. *Journal of Artificial Intelligence Research* **15**, 351–381.
5. Bernstein, D., R. Givan, N. Immerman, and S. Zilberstein: 2002, 'The Complexity of Decentralized Control of Markov Decision Processes'. *Mathematics of Operations Research* **27**(4), 819–840.
6. Bertsekas, D. and J. Tsitsiklis: 1996, *Neurodynamic Programming.* Athena Scientific.
7. Boutilier, C.: 1996, 'Planning, Learning and Coordination in Multiagent Decision Processes'. In: Y. Shoham (ed.): *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK '96).* pp. 195–210.
8. Buffet, O.: 2003, 'Une double approche modulaire de l'apprentissage par renforcement pour des agents intelligents adaptatifs'. Ph.D. thesis, Université Henri Poincaré, Nancy 1. Laboratoire Lorrain de recherche en informatique et ses applications (LORIA).
9. Buffet, O. and D. Aberdeen: 2006, 'The Factored Policy Gradient planner (IPC-06 Version)'. In: A. Gerevini, B. Bonet, and B. Givan (eds.): *Proceedings of the Fifth International Planning Competition (IPC-5).* pp. 69–71. [Winner, probabilistic track of the 5th International Planning Competition].
10. Buffet, O., A. Dutech, and F. Charpillet: 2004, 'Self-Growth of Basic Behaviors in an Action Selection Based Agent'. In: S. Schaal, A. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, and J.-A. Meyer (eds.): *From Animals to Animats 8: Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior (SAB'04).* pp. 223–232.
11. Buffet, O., A. Dutech, and F. Charpillet: 2005, 'Développement autonome des comportements de base d'un agent'. *Revue d'Intelligence Artificielle* **19**(4–5), 603–632.

12. Carmel, D. and S. Markovitch: 1996, *Adaption And Learning In Multi-Agent Systems*, Vol. 1042 of *Lecture Notes in Artificial Intelligence*, Chapt. Opponent Modeling in Multi-agent Systems, pp. 40–52. Springer-Verlag.
13. Cassandra, A. R.: 1998, 'Exact and Approximate Algorithms for Partially Observable Markov Decision Processes'. Ph.D. thesis, Brown University, Department of Computer Science, Providence, RI.
14. Dorigo, M. and G. Di Caro: 1999, 'Ant Colony Optimization: A New Meta-Heuristic'. In: P. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala (eds.): *Proceedings of the Congress on Evolutionary Computation (CEC-99)*. pp. 1470–1477.
15. Dutech, A.: 2000, 'Solving POMDP using selected past-events'. In: W. Horn (ed.): *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI'00)*. pp. 281–285.
16. Fernández, F. and L. Parker: 2001, 'Learning in Large Cooperative Multi-Robot Domains'. *International Journal of Robotics and Automation* **16**(4), 217–226.
17. Gerkey, B. and M. Matarić: 2004, 'A formal analysis and taxonomy of task allocation in multi-robot systems'. *International Journal of Robotics Research* **23**(9), 939–954.
18. Gmytrasiewicz, P. and P. Doshi: 2004, 'Interactive POMDPs: Properties and Preliminary Results'. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*.
19. Goldman, C., M. Allen, and S. Zilberstein: 2004, 'Decentralized Language Learning Through Acting'. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*.
20. Hengst, B.: 2002, 'Discovering Hierarchy in Reinforcement Learning with HEXQ'. In: C. Sammut and A. G. Hoffmann (eds.): *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*. pp. 243–250.
21. Hu, J. and M. Wellman: 1998, 'Online learning about other agents in a dynamic multiagent system'. In: K. P. Sycara and M. Wooldridge (eds.): *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*. pp. 239–246.
22. Jaakkola, T., M. Jordan, and S. Singh: 1994, 'On the Convergence of Stochastic Iterative Dynamic Programming Algorithms'. *Neural Computation* **6**(6), 1186–1201.
23. Jong, E. D.: 2000, 'Attractors in the Development of Communication'. In: J.-A. Meyer, A. Berthoz, D. Floreano, H. L. Roitblat, and S. W. Wilson (eds.): *From Animals to Animats 6: Proceedings of the 6th International Conference on Simulation of Adaptive Behavior (SAB-00)*.
24. Kaelbling, L., M. Littman, and A. Moore: 1996, 'Reinforcement Learning: A Survey'. *Journal of Artificial Intelligence Research* **4**, 237–285.
25. Laud, A.: 2004, 'Theory and Application of Reward Shaping in Reinforcement Learning'. Ph.D. thesis, University of Illinois at Urbana-Champaign.
26. Littman, M., A. Cassandra, and L. Kaelbling: 1995, 'Learning policies for partially observable environments: scaling up'. In: A. Prieditis and S. J. Russell (eds.): *Proceedings of the Twelveth International Conference on Machine Learning (ICML'95)*. pp. 362–370.
27. Matarić, M.: 1997, 'Reinforcement Learning in the Multi-Robot Domain'. *Autonomous Robots* **4(1)**, 73–83.
28. McCallum, R. A.: 1995, 'Reinforcement Learning with Selective Perception and Hidden State'. Ph.D. thesis, University of Rochester.

29. Ng, A., D. Harada, and S. Russell: 1999, 'Policy invariance under reward transformations: Theory and application to reward shaping'. In: I. Bratko and S. Dzeroski (eds.): *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*. pp. 278–287.
30. Peshkin, L., K. Kim, N. Meuleau, and L. Kaelbling: 2000, 'Learning to Cooperate via Policy Search'. In: C. Boutilier and M. Goldszmidt (eds.): *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI'00)*. pp. 489–496.
31. Peters, J., S. Vijayakumar, and S. Schaal: 2005, 'Natural Actor-Critic'. In: J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo (eds.): *Proceedings of the Sixteenth European Conference on Machine Learning (ECML'05)*, Vol. 3720 of *Lecture Notes in Computer Science*.
32. Puterman, M. L.: 1994, *Markov Decision Processes–Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., New York, USA.
33. Pynadath, D. and M. Tambe: 2002, 'The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models'. *Journal of Artificial Intelligence Research* **16**, 389–423.
34. Randløv, J.: 2000, 'Shaping in Reinforcement Learning by Changing the Physics of the Problem'. In: P. Langley (ed.): *Proceedings of the Seventeenth International Conference on Machine Learning, (ICML'00)*. pp. 767–774.
35. Randløv, J. and P. Alstrøm: 1998, 'Learning to Drive a Bicycle using Reinforcement Learning and Shaping'. In: J. W. Shavlik (ed.): *Proceedings of the Fifteenth International Conference on Machine Learning, (ICML'98)*. pp. 463–471.
36. Rogowski, C.: 2004, 'Model-based Opponent Modelling in Domains Beyond the Prisoner's Dilemma'. In: *Proceedings of Modeling Other Agents from Observations (MOO 2004), AAMAS'04 workshop*.
37. Salustowicz, R., M. Wiering, and J. Schmidhuber: 1998, 'Learning Team Strategies: Soccer Case Studies'. *Machine Learning* **33**, 263–282.
38. Shoham, Y., R. Powers, and T. Grenager: 2003, 'Multi-agent reinforcement learning: a critical survey'. Technical report, Stanford.
39. Singh, S., T. Jaakkola, and M. Jordan: 1994, 'Learning without state estimation in Partially Observable Markovian Decision Processes'. In: W. W. Cohen and H. Hirsh (eds.): *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*.
40. Skinner, B.: 1953, *Science and Human Behavior*. Collier-Macmillian, New-York.
41. Staddon, J.: 1983, *Adaptative Behavior and Learning*. Cambridge University Press.
42. Stone, P. and M. Veloso: 2000a, 'Layered Learning'. In: R. L. de Mántaras and E. Plaza (eds.): *Proceedings of the Eleventh European Conference on Machine Learning (ECML'00)*, Vol. 1810 of *Lecture Notes in Computer Science*.
43. Stone, P. and M. Veloso: 2000b, 'Multiagent systems: a survey from a machine learning perspective'. *Autonomous Robotics* **8**(3).
44. Stone, P. and M. Veloso: 2000c, 'Team-Partitioned, Opaque-Transition Reinforcement Learning'. In: *Proceedings of the Third International Conference on Autonomous Agents (Agents'00)*.
45. Sutton, R. and G. Barto: 1998, *Reinforcement Learning: an introduction*. Bradford Book, MIT Press, Cambridge, MA.
46. Sutton, R., D. McAllester, S. Singh, and Y. Mansour: 1999, 'Policy Gradient Methods for Reinforcement Learning with Function Approximation'. In: S. A.

Solla, T. K. Leen, and K.-R. Müller (eds.): *Advances in Neural Information Processing Systems 11 (NIPS'99)*, Vol. 12. pp. 1057–1063.
47. Tumer, K. and D. Wolpert: 2000, 'Collective Intelligence and Braess' Paradox'. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'00)*. pp. 104–109.
48. Tyrrell, T.: 1993, 'Computational Mechanisms for Action Selection'. Ph.D. thesis, University of Edinburgh.
49. Vidal, J. and E. Durfee: 1997, 'Agent learning about agents: a framework and analysis'. In: S. Sen (ed.): *Collected papers from the AAAI-97 workshop on multiagent learning*. pp. 71–76.
50. Watkins, C.: 1989, 'Learning from delayed rewards'. Ph.D. thesis, King's College of Cambridge, UK.
51. Wolpert, D. and K. Tumer: 1999, 'An introduction to collective intelligence'. Technical Report NASA-ARC-IC-99-63, NASA AMES Research Center.
52. Wolpert, D., K. Wheeler, and K. Tumer: 1999, 'General principles of learning-based multi-agent systems'. In: *Proceedings of the 3rd International Conference on Autonomous Agents (Agents'99)*. pp. 77–83.
53. Xuan, P., V. Lesser, and S. Zilberstein: 2000, 'Communication in Multi-Agent Markov Decision Processes'. In: S. Parsons and P. Gmytrasiewicz (eds.): *Proceedings of ICMAS Workshop on Game Theoretic and Decision Theoretic Agents*.

Address for Offprints: Address for correspondence