Actes des 13es Journées Francophones Planification, Décision, Apprentissage

Olivier Buffet

APIA & CNIA & IC & JFPDA & RJCIA

IA pour l'éducation IA & Santé TAL & IA Ethique & IA
Robotique & IA France@IJCAI2018

PFIA 2018

11e Plate-forme Intelligence Artificielle

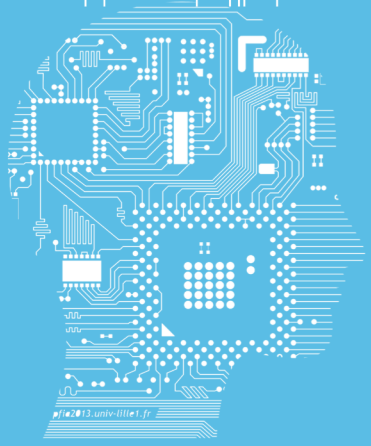
2 au 6 juillet 2018 - Nancy

Campus Sciences - Université de Lorraine
Vandœuvre-lès-Nancy

JFPDA 2018

Actes des Journées Francophones Planification, Décision, Apprentissage pour la conduite des systèmes

Président du comité de programme
Olivier Buffet



PFIA2018.LORIA.FR

"

Table des matières

Préambule

Ces actes présentent les articles soumis à l'édition 2018 des Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA), dans l'ordre de présentation initialement prévu.

Présentation

Les Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA) ont pour but de rassembler la communauté de chercheurs francophones travaillant sur les problèmes d'intelligence artificielle, d'apprentissage par renforcement, de programmation dynamique et plus généralement dans les domaines liés à la prise de décision séquentielle sous incertitude et à la planification. Les travaux présentés traitent aussi bien d'aspects purement théoriques que de l'application de ces méthodes à la conduite de systèmes virtuels (jeux, simulateurs) et réels (robots, drones). Ces journées sont aussi l'occasion de présenter des travaux en cours de la part de doctorants, postdoctorants et chercheurs confirmés dans un cadre laissant une large place à la discussion constructive et bienveillante.
Après Toulouse (2006), Grenoble (2007), Metz (2008), Paris (2009), Besançon (2010), Rouen (2011), Nancy (2012), Lille (2013), Liège (2014), Rennes (2015), Grenoble (2016) et Caen (2017), les 13es journées auront lieu à Nancy dans le cadre de la Plate-forme Intelligence Artificielle, les 2 et 3 juillet 2018.

Comité de programme

Co-Présidents

— Olivier Buffet - LORIA, INRIA
— Alain Dutech - LORIA, INRIA
— Vincent Thomas - LORIA, Université de Lorraine

Comité de programme

— Aurélie Beynier - LIP6, Université Pierre et Marie Curie (Paris 6)
— Martin Cooper - IRIT, Université Paul Sabatier
— Jilles Dibangoye - CITI, INSA-Lyon, Université de Lyon, INRIA
— Humbert Fiorino - LIG, Université Grenoble Alpes
— Matthieu Geist - LIEC, Université de Lorraine
— Laurent Jeanpierre - GREYC, Université Caen-Normandie
— Emilie Kaufmann - CRIStAL, CNRS, Lille
— Frédéric Maris - IRIT, Université Paul Sabatier - Toulouse III
— Marc Metivier - LIPADE, Université Paris Descartes (Paris 5)
— Abdel-Illah Mouaddib - GREYC, Université Caen-Normandie
— Damien Pellier - LIG, Université Grenoble Alpes
— Philippe Preux - CRIStAL, INRIA, Université de Lille
— Emmanuel Rachelson - ISAE-SUPAERO, Toulouse
— Bruno Zanuttini - GREYC, Université Caen-Normandie

Recherche heuristique pour jeux stochastiques (à somme nulle)

Olivier buffet[1] Jilles Dibangoye[2] Abdallah Saffidine[3] Vincent Thomas[1]

[1] Université de Lorraine, INRIA, CNRS, LORIA, F-54000 Nancy
[2] Université de Lyon, INSA Lyon and Inria, CITI, F-69000 Lyon
[3] Australian National University, Canberra, Australie

prenom.nom@(inria.fr|anu.edu.au)

Résumé

Dans divers types de problèmes, par exemple de prise de décision séquentielle, les algorithmes de recherche heuristique permettent d'exploiter la connaissance d'une situation initiale et d'une heuristique admissible pour rechercher efficacement une solution optimale. De tels algorithmes existent y compris en cas de dynamique incertaine, d'observabilité partielle, de critères multiples, ou d'agents multiples collaborant. Nous proposons ici un algorithme de recherche heuristique pour jeux stochastiques à deux joueurs et à somme nulle, et avec critère décompté, algorithme reposant sur HSVI—donc sur la génération de trajectoires. Nous démontrons que, chaque joueur agissant de manière optimiste, et en employant des initialisations heuristiques simples, l'algorithme obtenu converge vers une solution ϵ-optimale en temps fini.

Mots Clef

Recherche heuristique, jeux stochastiques, HSVI.

Abstract

In various types of problems, such as sequential decision-making, heuristic search algorithms allow exploiting the knowledge of the initial situation and of an admissible heuristic to efficiently search for an optimal solution. Such algorithms exist including in case of uncertain dynamics, of partial observability, of multiple criteria, or of multiple collaborating agents. Here we propose a heuristic search algorithm for two-player zero-sum stochastic games with discounted critrion. This algorithm relies on HSVI—hence on generating trajectories. We demonstrate that, each player acting in an optimistic manner, and employing simple heuristic initializations, the resulting algorithm converges in finite time to an ϵ-optimal solution.

Keywords

Heuristic Search, Stochastic Games, HSVI.

1 Introduction

Les techniques de recherche heuristiques ont été introduites pour résoudre des problèmes d'optimisation à un seul agent et un seul critère dans des cadres déterministes (souvent, mais pas seulement, à des fins de planification) avec des algorithmes tels que A*, (L)RTA*, ou RBFS [20, 7]. Elles ont été étendues pour résoudre des problèmes avec des dynamiques stochastiques dans les cadres MDP et POMDP comme dans (L)RTDP, (L)AO*, ou HSVI [5, 9, 25], et même dans des cadres multi-agents collaboratifs dans MAA* ou FB-HSVI [27, 8]. Dans tous ces cas, un seul critère est considéré, de sorte que guider l'exploration en étant optimiste reste une chose assez triviale à faire, même en planifiant pour plusieurs agents. Quand plusieurs critères sont considérés pour un seul agent, ou éventuellement des agents collaborants, ce guidage peut être adapté aussi bien quand on agrège ces critères que quand on cherche un front de Pareto [18]. À notre connaissance, la recherche heuristique n'a pas été appliquée aux cadres multi-agents non-collaboratifs avec jeu simultané, c'est-à-dire dans des jeux (stochastiques) où les joueurs ont leur propres objectifs.

Cet article se concentre sur les jeux stochastiques décomptés à deux joueurs et somme nulle (zs-SG) avec information parfaite (à l'exception du mouvement courant de l'adversaire), lesquels ont été principalement abordés en utilisant des algorithmes tels que la programmation dynamique exacte [17] (de manière similaire aux algorithmes d'itération sur la valeur ou la politique pour les MDP), l'apprentissage par renforcement [12], ou la programmation dynamique approchée [10, 16]. Le critère de la récompense totale, commun dans les jeux de plateau, requière que toutes les trajectoires finissent dans des états terminaux, et a été abordé avec l'apprentissage par renforcement [12], l'induction arrière [21, 6] (c'est-à-dire en applicant la programmation dynamique sur les états accessibles quand on élague les branches non pertinentes) et MCTS avec mouvements simultanés (Simultaneous Move MCTS) [6]. Pour d'autres critères, se référer à [26]. Les seuls algorithmes exacts (ou plus précisément à erreur bornée) listés ci-dessus sont ceux reposant sur la programmation dynamique. Parmi ceux-ci, l'induction arrière, parce qu'elle élague typiquement des branches en exploitant des estimations de valeurs conservatives, peut être vue comme une forme de recherche heu-

ristique.
Dans nos zs-SG décomptés, nous faisons l'hypothèse qu'un état initial s_0 est donné, et que des majorants et minorants initiaux (U et L) de la fonction de valeur optimale (qui serviront d'heuristiques admissibles) sont disponibles. Sous ces hypothèses, nous considérons des schémas algorithmiques apprenant en générant des trajectoires (à la LRTA*, (L)RTDP ou HSVI). Nous montrons que, dans ce cadre, les trajectoires peuvent être générées en laissant les deux joueurs agit de manière optimiste (l'un de manière gourmande par rapport au majorant $U(s,a)$ de la valeur optimale (en équilibre de Nash), l'autre par rapport au minorant $L(s,a)$) tout en assurant que ces encadrements vont converger vers l'optimum. Utiliser majorant et minorant amène naturellement vers des algorithmes ressemblant à Bounded-RTDP et HSVI [13, 25].
L'article est organisé comme suit. Il fournit d'abord un contexte sur les jeux en forme normale, les processus de décision markoviens (MDP), et les jeux stochastiques en section 2. Ensuite, quelques résultats préliminaires sur les jeux en forme normale sont présentés en section 3 quand on raisonne sur des jeux minorants ou majorants. Cela conduit à proposer les ingrédients clefs pour dériver une variante de l'algorithme HSVI (avec des preuves de convergence) en section 4. Finalement, la section 5 conclut avec une discussion sur de possibles travaux futurs.

2 Contexte

Dans ce qui suit, les exposants indiquent les joueurs, et les indices indiquent le pas de temps ou l'itération.

2.1 Jeux en forme normale/matricielle

Un jeu à deux joueurs en *forme normale* est défini par un tuple $\Gamma \stackrel{\text{def}}{=} \langle \mathcal{A}^1, \mathcal{A}^2, v^1, v^2 \rangle$, où $\mathcal{A}^i$ est l'ensemble fini des stratégies pures du joueur i ($i \in \{1,2\}$), et $v^i : \mathcal{A}^1 \times \mathcal{A}^2 \mapsto \mathbb{R}$ est la fonction de gain du joueur i. L'objectif est alors, pour chaque joueur, de maximiser son espérance de gain. À moins qu'ils aient des fonctions de gain identiques, il s'agit de trouver non pas l'optimum d'une fonction, mais u *équilibre*. Un concept solution classique est celui d'*équilibre de Nash* (NE) [15], défini comme une paire $(d^1, d^2) \in \Delta(\mathcal{A}^1) \times \Delta(\mathcal{A}^2)$ de stratégies mixtes (c'est-à-dire des distributions de probabilités sur les stratégies pures) telle qu'aucun joueur n'a intérêt de dévier de cette solution seul :

$$\forall d'^1 \in \Delta(\mathcal{A}^1), \quad v^1(d'^1, d^2) \leq v^1(d^1, d^2),$$
$$\forall d'^2 \in \Delta(\mathcal{A}^2), \quad v^2(d^1, d'^2) \leq v^2(d^1, d^2).$$

Dans un jeu à somme nulle, $v^1 + v^2 = 0$ (c'est-à-dire que le gain d'un joueur est la perte de l'autre), et nous noterons typiquement $v = v^1 = -v^2$. Aussi, comme démontré par von Neumann [28], un tel jeu Γ a une unique valeur d'équilibre de Nash $\text{NEV}(\Gamma)$ égale à la fois aux valeurs minimax et maximin. En conséquence, les stratégies formant une stratégie en équilibre de Nash (joint) (NES) peuvent être trouvées en résolvant (typiquement en tant que programmes linéaires) :

$$d^{NES,1} = \arg\max_{d^1 \in \Delta(\mathcal{A}^1)} \min_{a^2 \in \mathcal{A}^2} v(d^1, a^2),$$
$$d^{NES,2} = \arg\min_{d^2 \in \Delta(\mathcal{A}^2)} \max_{a^1 \in \mathcal{A}^1} v(a^1, d^2).$$

2.2 Jeux stochastiques

Un jeu stochastique (ou de Markov) décompté à deux joueurs et somme nulle (zs-SG) [22, 23, 17] est spécifié par un tuple $\langle \mathcal{S}, \mathcal{A}^1, \mathcal{A}^2, P, r, \gamma, s_0 \rangle$ où $\mathcal{S}$ est un ensemble fini d'états, $\mathcal{A}^1$ et $\mathcal{A}^2$ sont des ensembles finis d'actions (une par joueur), $P_{a^1,a^2}(s'|s)$ est la probabilité de transiter de l'état s à s' quand les actions a^1 et a^2 sont effectuées ; $r(s, a^1, a^2)$ est une fonction de récompense (scalaire) ; $\gamma \in [0, 1)$ est un facteur d'atténuation ; et s_0 est l'état initial. L'objectif du joueur 1 est de maximiser l'espérance de la somme atténuée des récompenses $E[\sum_t \gamma^t R_t | s_0]$ alors que l'objectif du joueur 2 est de minimiser cette quantité.
Par commodité, on notera :

- $\pi^i : \mathcal{S} \to \Delta(\mathcal{A}^i)$ une stratégie stochastique pour le joueur i ;
- $\pi = (\pi^1, \pi^2)$ une stratégie jointe (ou une paire de stratégies) pour les deux joueurs ;
- $\boldsymbol{r}(\pi)$ le vecteur des récompenses immédiates espérées pour la stratégie (jointe) π : $\boldsymbol{r}(\pi)(s) = \sum_{a^1,a^2} \pi^1(a^1|s)\pi^2(a^2|s)r(s, a^1, a^2)$;
- $P(\pi)$ la matrice de transition induite par la stratégie π.

Un tel jeu peut être ré-écrit en forme normale et résolu en cherchant alors un équilibre de Nash (une stratégie mixte). Toutefois, un concept solution plus satisfaisant est ici celui de *stratégie équilibre de Nash parfaite en sous-jeux* (stratégies SGPNE), c'est-à-dire de solutions consistant en un équilibre de Nash par état (rencontré). Définissons, pour tout $\boldsymbol{v}$, le *jeu de matrice de Shapley* $\Gamma^s(\boldsymbol{v}) \stackrel{\text{def}}{=} [r(s,\cdot,\cdot) + \gamma \sum_{s'} P_{\cdot,\cdot}(s'|s)\boldsymbol{v}(s')]$ pour tout s (c'est-à-dire, une valeur par paire d'action (a^1, a^2)). Alors, *l'opérateur d'optimalité de Shapley* $\mathcal{H} : \boldsymbol{v} \mapsto \text{NEV}(\Gamma^{\cdot}(\boldsymbol{v}))$ est une fonction contractante, et son unique point fixe est la fonction de valeur des stratégies SGPNE, notée NEV^*. *L'algorithme de Shapley* pour résoudre les zs-SG, analogue à *l'itération sur la valeur (VI)* pour les MDP, consiste en l'application itérative de cet opérateur jusqu'à ϵ-convergence, et alors, quand on est dans un état s, d'agir selon la NES du jeu de Matrice de Shapley induit en s. D'autres algorithmes exacts ont été proposés qui gèrent aussi des problèmes sans états initiaux connus [17].

2.3 MDP et HSVI

Les processus de décision markoviens (MDP) [2, 3] peuvent être vus comme des jeux stochastiques dans lesquels un joueur (sans perte de généralité, le joueur minimiseur 2) a une seule action disponible, et donc aucune décision à prendre. Le problème est alors de trouver une stratégie (aussi appelée *politique*) pour le joueur

maximiseur 1. Ici la NEV devient la *fonction de valeur optimale* V^*, et agir de manière gloutonne par rapport à cette fonction de valeur induit une politique optimale, de sorte qu'il existe toujours une politique optimale déterministe. Par commodité, pour une fonction de valeur V, nous introduisons la Q^V-function correspondante $Q^V(s,a) \stackrel{\text{def}}{=} r(s,a) + \gamma \sum_{s'} P_a(s'|s)V(s')$.

Bounded RTDP [13] et HSVI [25] sont deux algorithmes résolvant des MDP en reposant sur (i) un état initial s_0 pour se focaliser sur les parties les plus pertinentes de l'espace d'états, (ii) des majorants et minorants (U et L) de la fonction de valeur optimale V^*, et (iii) des mises à jour ponctuelles de ces encadrements en les états rencontrés en suivant des trajectoires. Alors que HSVI a été initialement introduit pour résoudre des POMDP, nous nous concentrons sur sa version générique (MDP) étudiée dans [24] (présentée dans l'algorithme 1) plutôt que BRTDP, parce qu'il vient avec des garanties théoriques plus fortes (par exemple, il converge en temps fini plutôt qu'à la limite). HSVI et BRTDP sélectionnent tous deux des actions de manière gourmande par rapport au majorant, et arrêtent d'échantillonner des trajectoires quand l'écart $U(s_0) - L(s_0)$ est sous un seuil $\epsilon > 0$. Pour sélectionner le prochain état étant donnés s et a, HSVI choisi un état s' maximisant $excess(s') = P_a(s'|s)(U(s') - L(s') - \gamma^{-\delta}\epsilon)$, où δ est la profondeur de s' dans la trajectoire courante (de manière à se focaliser sur les états dont les mises à jours vont plus vraisemblablement aider). [1]

Algorithme 1 : Heuristic Search Value Iteration

```
1  Fct HSVI (ε)
2      Initialize L and U
3      while (U(s_0) − L(s_0)) > ε do
4          RecursivelyTry (s_0, δ = 0)
5      return L
6  Fct RecursivelyTry (s, δ)
7      if (U(s) − L(s)) > γ^{−δ} ε then
8          Update (s)
9          a* ∈ arg max_{a∈A} Q^U(s, a)
10         s' ∈ arg max_{s"∈S} Pr(s"|s, a*)
             × (U(s") − L(s") − γ^{−δ} ε)
11         RecursivelyTry (s', δ + 1)
12         Update (s)
13     return
14 Fct Update (s)
15     L ← Update (L, s)
16     U ← Update (U, s)
```

1. BRTDP échantillone un état selon une distribution qui ressemble à une version normalisée de cette fonction excès.

3 Encadrer un jeu en forme normale

Comme expliqué dans l'introduction, nous souhaiterions proposer des algorithmes de recherche heuristique (à la HSVI) pour des zs-SG à 2 joueurs, c'est-à-dire en concentrant l'effort computationel sur les parties pertinentes de l'espace d'état en exploitant (i) la connaissance de l'état initial s_0, et (ii) des heuristiques admissibles pour les deux joueurs, qui seront employées pour initialiser le majorant et le minorant de la fonction de valeur. Alors que la recherche heuristique classique (1 joueur contre la nature) requière seulement un majorant de la fonction de valeur pour guider la recherche de manière optimiste, les zs-SG à 2 joueurs requièrent à la fois un majorant et un minorant (de manière à être optimiste pour les deux joueurs). Nous allons donc calculer des équilibres de Nash (NE) à la fois pour les jeux de matrices de Shapley majorant $\Gamma^s(U)$ et minorant $\Gamma^s(L)$ dans chaque état rencontré s. Cela soulève une première question : Que se passe-t-il si, dans un état s, les deux NE sont égaux ? Comme démontré par les deux résultats suivants, dans ce cas la valeur obtenue est la valeur (SGP)NE pour cet état.

Lemme 1 *Si le gain v^{up} du jeu à somme nulle en forme normale Γ^{up} majore le gain v^{lo} du jeu à somme nulle en forme normale (et de même dimensions) Γ^{lo}, alors* $\text{NEV}(\Gamma^{up}) \geq \text{NEV}(\Gamma^{lo})$.

Preuve 1

$$\forall a^1, a^2 \quad v^{up}(a^1,a^2) \geq v^{lo}(a^1,a^2)$$

$$\forall a^1 \quad \min_{a^2} v^{up}(a^1,a^2) \geq \min_{a^2} v^{lo}(a^1,a^2)$$

$$\max_{d^1} \min_{a^2} \sum_{a^1} d^1(a^1) v^{up}(a^1,a^2) \geq \max_{d^1} \min_{a^2} \sum_{a^1} d^1(a^1) v^{lo}(a^1,a^2)$$

c'est-à-dire, $\text{NEV}(\Gamma^{up}) \geq \text{NEV}(\Gamma^{lo})$. □

Corollaire 1 *Dans un jeu stochastique à somme nulle (zs-SG), si, dans un état s, les valeurs NE de $\Gamma^s(L)$ et $\Gamma^s(U)$ sont égales, alors* $\text{NEV}(\Gamma^s(L)) = \text{NEV}(\Gamma^s(U))$ *est la valeur NE,* $\text{NEV}^*(s)$*, du jeu de Shapley à somme nulle obtenu après convergence dans cet état (pour le SGPNE de ce zs-SG).*

Preuve 2 *En appliquant le lemme précédent dans un état s, on a toujours dans notre cadre :* $\text{NEV}(\Gamma^s(L)) \leq \text{NEV}^*(s) \leq \text{NEV}(\Gamma^s(U))$*. Ainsi, quand* $\text{NEV}(\Gamma^s(L)) = \text{NEV}(\Gamma^s(U))$*, c'est aussi la valeur de* $\text{NEV}^*(s)$. □

La section suivante discute de la façon de dériver une variante de HSVI pour les jeux stochastiques à deux joueurs et somme nulle.

4 Algorithme et preuve de convergence

Cette section montre comment adapter HSVI au cadre zs-SG. Tant que $\text{NEV}(\Gamma^{s_0}(U)) - \text{NEV}(\Gamma^{s_0}(L)) > \epsilon$, HSVI

génère des trajectoires en appliquant, dans chaque état rencontré, (i) une règle de sélection d'action et (ii) une règle de sélection du prochain état.[2] Nous présentons maintenant ces règles et d'autres détails de l'algorithme, et prouvons qu'il trouve des stratégies ϵ-optimale en s_0 en temps fini.

4.1 Sélection de l'action jointe

Dans le cadre mono-joueur classique, agir de manière gourmande par rapport au majorant (une estimation optimiste) est une règle de sélection d'action valide pour garantir la convergence. Dans un zs-SG, une estimation optimiste pour le joueur 1 est pessimiste pour le joueur 2. Supposant que le joueur 1 n'a qu'une seule action possible, agir de manière gourmande par rapport au majorant est l'opposé de ce que devrait faire le joueur 2.
La règle d'exploration proposée est, étant dans l'état s,

— le joueur 1 choisit l'action stochastique $d^{U,1}$ de manière gourmande par rapport au majorant U, c'est-à-dire en jouant au jeu de matrice à somme nulle de Shapley $\Gamma^s(U)$; et
— le joueur 2 choisit l'action stochastique $d^{L,2}$ de manière gourmande par rapport au minorant L, c'est-à-dire en jouant au jeu de matrice à somme nulle de Shapley $\Gamma^s(L)$.

On appelle la règle de décision jointe résultante $d = (d^{U,1}, d^{L,2})$ une *tentative de NES* en s. La figure 1 représente la matrice de jeu pour un état s avec les actions impliquées dans les stratégies en NE $\text{NES}(\Gamma^s(U))$ et $\text{NES}(\Gamma^s(L))$ qui ont été trouvées.

4.2 Sélection du prochain état

Après avoir choisit une stratégie jointe, le prochain état s' peut être sélectionné de différentes manières (par exemple, en échantillonnant a^1 de $d^1(\cdot)$, a^2 de $d^2(\cdot)$, puis s' de $P_{a^1,a^2}(\cdot|s)$ (à la (L)RTDP). Toutefois, pour préserver les garanties de convergence de HSVI, nous préférons sélectionner un prochain état s' maximisant

$$\sum_{a^1,a^2} d^1(a^1)d^2(a^2)P_{a^1,a^2}(s'|s) \underbrace{\left[U(s') - L(s') - \gamma^{-\delta}\epsilon\right]}_{excess(s',\delta)},$$

où δ est la profondeur courante de la trajectoire.

4.3 Mise à jour de la valeur

Comme pour HSVI classique, nous utilisons des opérateurs *à base de points* pour mettre à jour les minorants et majorants (c'est-à-dire des opérateurs qui sont appliqués en des points particuliers s), et définissons :

— L un minorant *uniformément améliorable* (UI) comme vérifiant $\mathcal{H}L \geq L$; et
— $\mathcal{K}^L_\cdot$ un opérateur de mise à jour à base de point *fort* (*strong*) pour le minorant comme vérifiant, pour chaque s où il est appliqué et chaque L, (i) $(\mathcal{H}L)(s) = (\mathcal{K}^L_s L)(s)$ et (ii) $(\mathcal{H}L)(s') \geq (\mathcal{K}^L_s L)(s')$ dans tout autre point s'

2. Nous utilisons le terme « règle » plutôt que « stratégie » pour éviter de possibles confusions.

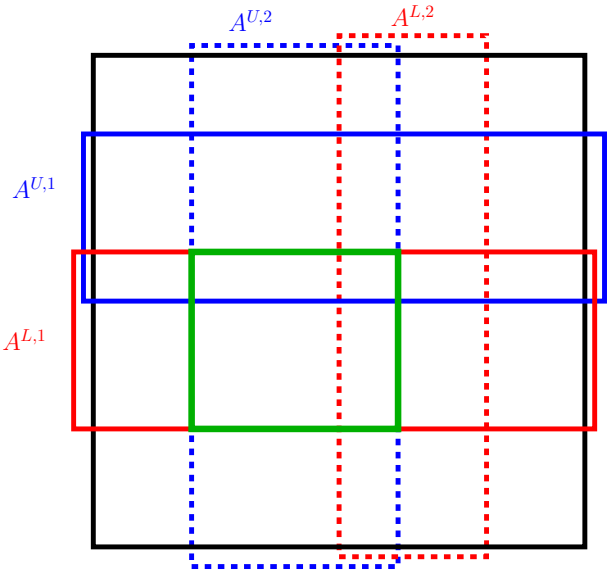


FIGURE 1 – La matrice de jeu pour un état s, avec les ensembles d'actions apparaissant dans les NES trouvées pour les jeux $\Gamma^s(U)$ et $\Gamma^s(L)$. Notes : (i) on suppose les actions ordonnées de manière à ce que celles intervenant dans les équilibres de Nash calculés soient regroupées ; (ii) le carré vert indique les actions jointes qui sont considérées par l'exploration ; (iii) les ensembles $A^{X,i}$ ne contiennent pas toutes les actions qui pourraient apparaître dans les NES.

(les mêmes définitions s'appliquent pour le majorant U en inversant les opérateurs d'inégalité). L'améliorabilité uniforme est nécessaire pour garantir que le minorant (resp. le majorant) reste un minorant (resp. un majorant). La propriété forte est requise pour garantir la convergence vers la fonction optimale à la limite.
Dans notre cadre, nous utilisons pour le minorant comme pour le majorant des représentations tabulaires et un même opérateur de mise à jour $\mathcal{K}_s$, lequel, pour tout état s, ne met à jour que la valeur pour cet état, et ce, avec les mises à jour naturelles :

$$(\mathcal{K}_s L)(s) \stackrel{\text{def}}{=} \text{NEV}(\Gamma^s(L)),$$
$$(\mathcal{K}_s U)(s) \stackrel{\text{def}}{=} \text{NEV}(\Gamma^s(U)),$$

et laisse L et U inchangés pour les autres états.
Comme dans le cadre MDP : (i) l'opérateur proposé est fort, donc conservatif [24, Def. 3.24, 3.25] ; et (ii) tout opérateur de mise-à-jour conservatif préserve l'améliorabilité uniforme (UI) [24, Th. 3.29]. Il nous reste donc essentiellement à nous assurer que nous employons des initialisations induisant des encadrements UI (voir prochaine sous-section).

4.4 Majorants et minorants

Comme pour les POMDP, une façon d'initialiser les minorants et majorants est de calculer le point fixe d'un opérateur qui minore ou majore l'opérateur d'optimalité de Shapley $\mathcal{H}$. La table 1 montre des opérateurs candidats pour

les deux encadrements, lesquels nous décrivons aussi verbalement comme suit :

- $\mathcal{G}_{\text{SEQ}}^{\cdot}$ **:** une approximation par jeu séquentiel—aussi appelée *sérialisation*—, c'est-à-dire U étant la valeur optimale if le joueur 1 agit avant le joueur 2, et L l'opposé (le second joueur sachant le choix d'action du premier) ;
- $\mathcal{G}_{\text{MDP}}^{\cdot}$ **:** une approximation MDP, c'est-à-dire U (respectivement L) étant la fonction de valeur optimale du MDP induit en affectant au joueur 2 (resp. au joueur 1) n'importe quelle stratégie fixe (par exemple des actions aléatoires) ;
- $\mathcal{G}_{\text{ORA}}^{\cdot}$ **:** une approximation MDP avec un oracle qui prédit l'action de l'adversaire randomisé (mais pas le mouvement de la nature) ; ou
- $\mathcal{G}_{\text{TRIV}}^{\cdot}$ **:** des approximations triviales, par exemple, $\forall s$, $U(s) = \frac{1}{1-\gamma}\max_{s',a^1,a^2} r(s',a^1,a^2)$ et $L(s) = \frac{1}{1-\gamma}\min_{s',a^1,a^2} r(s',a^1,a^2)$.

Dans la table 1, les opérateurs au sein d'une même boîte retourne des fonctions « plus grandes ou égales » à celles des opérateurs dans une boîte plus basse. À l'intérieur de chacune des deux boîtes contenant trois opérateurs, les seules relations d'ordre connues sont : $\mathcal{G}^U_{\text{ORA}}\boldsymbol{v} \geq \mathcal{G}^U_{\text{MDP}}\boldsymbol{v}$ et $\mathcal{G}^L_{\text{MDP}}\boldsymbol{v} \geq \mathcal{G}^L_{\text{ORA}}\boldsymbol{v}$. Évidemment, un problème est de savoir si dériver ces fonctions encadrantes en vaut l'effort de calcul. Les points fixes des opérateurs $\mathcal{G}^{\cdot}_{\text{TRIV}}$ peuvent être calculés en temps constant (voir formules ci-dessus), et servir d'initialisations pour calculer les points fixes des autres opérateurs, par exemple, à nouveau par recherche heuristique. Les opérateurs $\mathcal{G}^{\cdot}_{\text{ORA}}$ ne méritent pas d'être considérés parce qu'ils fourniraient de moins bons encadrements que d'autres opérateurs de même coût de calcul.

Améliorabilité uniforme Sans perte de généralité, discutons de l'améliorabilité uniforme (UI) seulement pour les majorants. Comme proposé ci-dessus, la majorant initial U est obtenu comme l'unique point fixe d'un opérateur de mise-à-jour $\mathcal{K}^{up}$ qui « majore » toujours l'opérateur d'optimalité de Shapley $\mathcal{H}$. En conséquence, $\mathcal{H}U \leq \mathcal{K}^{up}U = U$, de sorte que U est uniformément améliorable. Cette approche est similaire au théorème 3.20 dans [24, page 71], et permet aussi de démontrer que les minorants proposés sont UI.

Politique par défaut Finalement, nous allons voir en discutant de la politique à exécuter effectivement, que le U initial (resp. L) doit venir avec une stratégie pour le joueur 2 (resp. pour le joueur 1) de manière à assurer que le pire résultat associé avec cette fonction de valeur encadrante est garantie (ce qui est le cas pour les trois approches proposées ci-dessus). Cette exigence nous empêche d'exploiter les NEV nulles des états auto-symétriques dans les zs-SG symétriques (c'est-à-dire quand les situations des deux joueurs sont en miroir l'une de l'autre). [3]

3. On pourrait éventuellement exploiter le fait que, si s et s' sont des états symétriques, alors $U(s) = -L(s')$.

TABLE 1 – Divers opérateurs de mise à jour pour la fonction de valeur, *presque* ordonnés selon les valeurs résultantes. Au milieu se trouve l'opérateur de Shapley optimal. Au dessus se trouvent les opérateurs pour l'initialisation du majorant U. En dessous se trouvent les opérateurs pour l'initialisation du minorant L.

$(\mathcal{G}^U_{\text{TRIV}}\boldsymbol{v})(s)$	$=$	$\max_{s',a^1,a^2,s"} r(s',a^1,a^2)+\gamma\boldsymbol{v}(s")$	
$(\mathcal{G}^U_{\text{ORA}}\boldsymbol{v})(s)$	$=$	$\mathbb{E}_{a^2\sim\text{Unif}(\mathcal{A}^2)}\max_{a^1}$	$\Gamma^s(\boldsymbol{v})(a^1,a^2)$
$(\mathcal{G}^U_{\text{MDP}}\boldsymbol{v})(s)$	$=$	$max_{a^1}\mathbb{E}_{a^2\sim\text{Unif}(\mathcal{A}^2)}$	$\Gamma^s(\boldsymbol{v})(a^1,a^2)$
$(\mathcal{G}^U_{\text{SEQ}}\boldsymbol{v})(s)$	$=$	$\min_{a^2}\max_{a^1}$	$\Gamma^s(\boldsymbol{v})(a^1,a^2)$
$(\mathcal{H}\boldsymbol{v})(s)$	$=$	$\min_{d^2}\max_{a^1}$	$\Gamma^s(\boldsymbol{v})(a^1,d^2)$
$=\text{NEV}(\Gamma^s(\boldsymbol{v}))$	$=$	$\max_{d^1}\min_{a^2}$	$\Gamma^s(\boldsymbol{v})(d^1,a^2)$
$(\mathcal{G}^L_{\text{SEQ}}\boldsymbol{v})(s)$	$=$	$\max_{a^1}\min_{a^2}$	$\Gamma^s(\boldsymbol{v})(a^1,a^2)$
$(\mathcal{G}^L_{\text{MDP}}\boldsymbol{v})(s)$	$=$	$\min_{a^2}\mathbb{E}_{a^1\sim\text{Unif}(\mathcal{A}^1)}$	$\Gamma^s(\boldsymbol{v})(a^1,a^2)$
$(\mathcal{G}^L_{\text{ORA}}\boldsymbol{v})(s)$	$=$	$\mathbb{E}_{a^1\sim\text{Unif}(\mathcal{A}^1)}\min_{a^2}$	$\Gamma^s(\boldsymbol{v})(a^1,a^2)$
$(\mathcal{G}^L_{\text{TRIV}}\boldsymbol{v})(s)$	$=$	$\min_{s',a^1,a^2,s"} r(s',a^1,a^2)+\gamma\boldsymbol{v}(s")$	

4.5 Critère de terminaison des trajectoires et preuve de convergence

Interrompons une trajectoire quand on atteint un état s à profondeur δ s'il est *NEV-fini*, c'est-à-dire quand l'excès, $excess(s,\delta) \stackrel{\text{def}}{=} U(s) - L(s) - \gamma^{-\delta}\epsilon$, est négatif ou nul. L'utilisation de ce critère de terminaison des trajectoires est le dernier élément nécessaire pour garantir que la version zs-SG de HSVI converge en temps fini. Nous avons essentiellement besoin de prouver le lemme clef suivant avant d'exploiter des résultats de [24].

Lemme 2 (Cf. *[24, Lemme 6.1]*)
Soient $\mathcal{K}$. un opérateur de mise-à-jour fort, L et U des fonctions de valeurs uniformément améliorables, s un état, et $d^* = (d^{U,1}, d^{L,2})$ la NES tentative courante en s. Alors

$$width(\mathcal{K}_s\hat{V}(s)) \leq \gamma \sum_{s'} P_{d^{U,1},d^{L,2}}(s'|s)\, width(\hat{V}(s')),$$

où $\hat{V}$ désigne la paire (L,U) et $P_{\cdot,\cdot}$ est naturellement étendue pour les stratégies mixtes (/actions stochastiques).

Preuve 3 *Nous avons*

$$\begin{aligned}
&width(\mathcal{K}_s\hat{V}(s)) \\
&= width(\mathcal{H}\hat{V}(s)) && (\mathcal{K}_s \textit{ est fort}) \\
&= \mathcal{H}U(s) - \mathcal{H}L(s) && (\textit{déf. de } width(\cdot)) \\
&= \text{NEV}(\Gamma^s(U)) - \text{NEV}(\Gamma^s(L)) && (\textit{déf. de } \mathcal{H}) \\
&= \max_{d^1}\min_{a^2} v^U(s,d^1,a^2) - \min_{d^2}\max_{a^1} v^L(s,d^1,a^2) \\
&\leq v^U(s,d^{U,1},d^{L,2}) - v^L(s,d^{U,1},d^{L,2})
\end{aligned}$$

$$= \gamma \sum_{s'} P_{d^{U,1},d^{L,2}}(s'|s)(U(s') - L(s'))$$
$$= \gamma \sum_{s'} P_{d^{U,1},d^{L,2}}(s'|s) width(\hat{V}(s')). \qquad \square$$

Ce résultat est important parce qu'il majore la largeur en s après mise-à-jour par une combinaison linéaire des largeurs des états immédiatement atteignables depuis s avec d^*. Ainsi, on peut réduire la largeur en s si l'on est capable de réduire suffisamment les largeurs de ces états suivants (dont la liste peut changer d'une visite de s à l'autre).

Corollaire 2 *La variante de HSVI pour zs-SG spécifiée par les processus de sélection, les opérateurs de mise-à-jour, et le critère de terminaison de trajectoires ci-dessus converge en temps fini.*

Preuve 4 *(ébauche inspirée de [24]) D'abord, le lemme précédent reste valable si l'on remplace la largeur (width) par l'excès (excess) [24, Lemma 6.2]. En conséquence, si tous les successeurs s' de s atteignables par d^* sont NEV-finis, alors s' est aussi NEV-fini [24, Lem. 6.3]. Aussi, HSVI sélectionne toujours un successeur non NEV-fini s'il en existe un [24, Lem. 6.4], et tous les états au-delà de la profondeur $\delta_{\max} \stackrel{\text{def}}{=} \lceil \log_\gamma(\epsilon/\|U - L\|_\infty) \rceil$ sont NEV-finis [24, Lem. 6.4] (par exemple, avec U_0 et L_0 les encadrements initiaux). Une conséquence est que toute trajectoire se termine et l'avant-dernier état visité devient NEV-fini [24, Lem. 6.7]. De là, l'arbre des états atteignables depuis s_0 a une profondeur bornée $\delta_{\max}$ et un facteur de branchement fini β, de sorte que s_0 sera NEV-fini (à profondeur 0) après au plus $\frac{\beta^{\delta_{\max}}-1}{\beta-1}$ trajectoires.* $\square$

L'algorithme complet est détaillé dans l'algorithme 2.

4.6 Exécution de la politique

Dans le cadre HSVI (ou BRTDP), un joueur doit agir de manière gourmande par rapport (i) à une estimation optimiste de la fonction de valeur optimale pour que la recherche heuristique converge (*cf.* la sélection d'action présentée ci-dessus), et (ii) à une estimation pessimiste lors de l'exécution de manière à garantir le pire retour espéré. Plus précisément, lors de l'exécution, si l'on est dans un état s, le joueur 1 (resp. 2) doit trouver une stratégie en équilibre de Nash d^L pour $\Gamma^s(L)$ (resp. d^U pour $\Gamma^s(U)$) et agir suivant $d^{L,1}$ (resp. $d^{U,2}$). Ainsi, le joueur 1 s'assure d'obtenir la valeur du « niveau de sécurité » associée à $\Gamma^s(L)$. L'exemple suivant illustre pourquoi agir de manière optimiste lors de l'exécution peut être préjudiciable.

Exemple 1 *Prenons le jeu de matrix Γ^α décrit en table 2 avec ses majorant et minorant U et L. U et L ont la même NEV, 0. Toutefois, selon U, n'importe quelle stratégie $(p, 1-p)$ du joueur (ligne) 1 ($p \in [0,1]$) fait partie d'une NES. Si le joueur 1 joue $(p, 1-p)$ et le joueur 2 joue $(0,1)$ (par exemple, parce que ce dernier sait qu'α est négatif), alors la valeur obtenue est $p\alpha$ (< 0), ce qui représente une perte pour le joueur 1.*

Algorithme 2 : zsSG-HSVI

```
Fct HSVI (ε)
    Initialize L and U
    while (U(s_0) − L(s_0)) > ε do
        RecursivelyTry (s_0, δ = 0)
    return L
Fct RecursivelyTry (s, δ)
    if (U(s) − L(s)) > γ^{−δ} ε then
        Update (s)
        d^U ← NES(Γ^s(U))
        d^L ← NES(Γ^s(L))
        s' ∈ arg max_{σ∈S} Σ_{a^1,a^2} d^{U,1}(a^1) d^{L,2}(a^2)
          × T(s, a^1, a^2, σ) [U(σ) − L(σ) − γ^{−δ} ε]
        RecursivelyTry (s', δ + 1)
        Update (s)
    return
Fct Update (s)
    L ← Update (L, s) /* uses NEV(Γ^s(L)) */
    U ← Update (U, s) /* uses NEV(Γ^s(U)) */
```

TABLE 2 – Un exemple de jeu Γ^α (où $\alpha \in [-1, +1]$) avec des exemples de jeux majorant et minorant U et L, leurs NES et leurs valeurs (à l'équilibre de Nash). Le joueur 1 est le joueur ligne.

jeu	matrice	NES	valeur
U	$\begin{bmatrix} 0 & +1 \\ 0 & 0 \end{bmatrix}$	$((p, 1-p), (1, 0))$	0
Γ_α	$\begin{bmatrix} 0 & \alpha \\ 0 & 0 \end{bmatrix}$	[voir texte]	0
L	$\begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix}$	$((0, 1), (q, 1-q))$	0

Notes :

- *Cet exemple montre aussi que l'égalité des NEV de U et V ne nécessite pas leur égalité sur le support de la tentative d'équilibre de Nash (c'est-à-dire sur les valeurs de la matrice impliquées).*
- *Γ^α a toujours une NEV de 0, mais ses NES possibles dépendent du signe ou de la nullité d'α.*

L'initialisation de L devrait ainsi venir avec une stratégie par défaut pour le joueur 1 (pré-calculée ou calculable en ligne) de manière à garantir que le joueur 1 a une action appropriée à effectuer même dans les états non rencontrés par l'algorithme (et de même pour U et le joueur 2).

5 Discussion

Cet article propose un algorithme de recherche heuristique pour les jeux stochastiques à (deux joueurs et) somme nulle

avec un état intial. Les critères opposés des joueurs amène naturellement à des algorithmes avec majorant et minorant tels que HSVI et BRTDP. Le principe de l'optimisme face à l'incertain est maintenu, ainsi que celui d'agir (lors de la phase d'exécution) suivant l'estimation pessimiste. Comme le HSVI original, il est prouvé que l'algorithme obtenu converge vers une solution ϵ-optimale en temps fini.
Nous énumérons maintenant quelques directions pour des travaux futurs.

Expérimentations zs-HSVI a encore besoin d'être implémenté et évalué empiriquement sur des bancs d'essai tels que un jeu de football à deux joueurs sur plateau [11, 10] ; le problème de contrôle de flux routeur/serveur [1, 10] ; ou des Garnets (c'est-à-dire des problèmes générés automatiquement) [16]. D'autres jeux de plateau pourraient être considérés, tels qu'Alesia [14] ou Goofspiel/GOPS [19], même si leurs critères ne sont habituellement pas atténués. L'analyse devrait comparer (i) zs-HSVI avec d'autres solveurs de zs-SG, et (ii) différentes méthodes d'initialisation des majorant et minorant.

Améliorations D'abord, il est commun d'améliorer le comportement anytime de HSVI en appelant la procédure de génération de trajectoires avec un paramètre ϵ de calcul de l'écart égal à une grande fraction de la largeur courant en s_0, par exemple $0,9(U(s_0) - L(s_0))$. Cela empêche les premières trajectoires d'être trop longues, et ressemble aux approches par approfondissement itérative (*iterative deepening*).
Par ailleurs, un problème est que de nombreux programmes linéaires (LP), potentiellement grands, devront être résolus. Comme dans DO$\alpha\beta$ [6], une première idée est de réduire les coûts de calcul en utilisant des méthodes avec oracles, c'est-à-dire des méthodes itératives qui permettent de résoudre de grands jeux en forme normale sans considérer toutes les stratégies possibles. [4]
Une autre idée est d'exploiter le fait que, quand un état est revisité, les LP qui doivent être résolus seront souvent très similaires à ceux qui ont déjà été résolus lors de la dernière visite. Cela suggère d'employer des techniques de *bootstrapping* (amorçage), c'est-à-dire d'initialiser les solveurs LP (si possible avec double oracle) en utilisant les dernières solutions.

Autre schémas algorithmiques Nous nous sommes concentrés jusqu'ici sur un algorithme reposant sur HSVI. Il serait intéressant de regarder si d'autres schémas algorithmiques pour la recherche heuristique, tels que LAO*, RTDP et HDP, pourraient être employés pour des jeux stochastiques. Comme souligné par Bonet et Geffner [4], tous les trois correspondent au schéma cherche-et-RÉVISE (FIND-and-REVISE), de sorte que les principaux résultats de convergence devraient pouvoir être abordés conjointement.

4. Des techniques apparentées ont été utilisées pour améliorer la procédure d'élagage dans les POMDP [29].

Problèmes sans atténuation Le présent travail fait l'hypothèse qu'un critère atténué est utilisé (comme requis par le HSVI de base). Une question qui se pose naturellement est comment adapter ce travail à un critère non atténué : Quels schémas algorithmiques pourraient être employés ? Sous quelles hypothèses ?

SG à somme générale Une direction pour des recherches futures est l'extension du présent travail à la résolution de jeux stochastiques à somme générale (avec si possible plus de deux joueurs). On peut s'attendre à ce que cela implique le maintient de majorants et minorants pour les critères de chaque joueur. Mais un premier problème est de choisir un concept solution approprié, puisqu'un jeu matriciel donné n'a plus nécessairement une valeur d'équilibre de Nash unique. Si on planifie pour tous les joueurs en même temps (en faisant l'hypothèse que chacun s'engagera à suivre la stratégie qui lui est affectée), alors n'importe quel NE peut être une solution valide. Mais que faire si on planifie pour un seul joueur sans avoir de contraintes sur les stratégies des autres joueurs ?

POSG / Occupancy zs-SG Un objectif à plus long terme de ce travail est la résolution de jeux stochastiques partiellement observables (POSG) en utilisant une variante de HSVI et une approximation de la fonction de valeur comme cela a été fait pour les Dec-POMDP [8], c'est-à-dire dans le cas complètement collaboratif. Dans le cas d'un POSG à 2 joueurs et somme nulle, cela supposerait de résoudre le problème comme un *occupancy zs-SG*, où l'état d'occupation est une statistique suffisante pour le planificateur. Le problème de planification devient alors déterministe, mais l'espace d'états est un simplexe (un espace de distributions de probabilités). Dans les occupancy MDP induits par des Dec-POMDP, le facteur de branchement reste fini parce que seules des règles de décision déterministes sont considérées, lesquelles sont en nombre fini. À l'inverse, dans les occupancy zs-SG, des règles de décision stochastiques doivent être considérées, lesquelles sont en nombre infini. On pourrait vraisemblablement obtenir une convergence ϵ-optimale en temps fini en utilisant un partitionnement récursif (qui induirait une discrétisation de l'espace d'actions, et donc de l'espace des états) si des approximations Lipschitz-continues (plutôt que PWLC) de la fonction de valeur peuvent être employées (comme nous le supposons pour l'instant).

Références

[1] E. Altman : Flow control using the theory of zero-sum Markov games. *IEEE Trans. on Auto. Control*, (39), 1994.

[2] R. Bellman : The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6), 1954.

[3] D. Bertsekas et J. Tsitsiklis : *Neurodynamic Programming*. 1996.

[4] B. Bonet et H. Geffner : Faster heuristic search algorithms for planning with uncertainty and full feedback. Dans *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

[5] B. Bonet et H. Geffner : Labeled RTDP : Improving the convergence of real-time dynamic programming. Dans *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS'03)*, 2003.

[6] B. Bošanský, V. Lisý, M. Lanctot, J. Čermák et M. H. Winands : Algorithms for computing strategies in two-player simultaneous move games. *Artificial Intelligence*, 237, 2016.

[7] V. Bulitko et G. Lee : Learning in real-time search : A unifying framework. *Journal of Artificial Intelligence Research (JAIR)*, 25, 2006.

[8] J. Dibangoye, C. Amato, O. Buffet et F. Charpillet : Optimally solving Dec-POMDPs as continuous-state MDPs. *Journal of Artificial Intelligence Research*, 55, 2016.

[9] E. Hansen et S. Zilberstein : LAO* : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129, 2001.

[10] M. G. Lagoudakis et R. Parr : Value function approximation inf zero-sum Markov games. Dans *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI'02)*, 2002.

[11] M. Littman : Markov games as a framework for multi-agent reinforcement learning. Dans *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, 1994.

[12] M. Littman et C. Szepesvári : A generalized reinforcement learning model : Convergence and applications. Dans *Proceedings of the International Conference on Machine Learning (ICML'96)*, 1996.

[13] H. B. McMahan, M. Likhachev et G. J. Gordon : Bounded real-time dynamic programming : RTDP with monotone upper bounds and performance guarantees. Dans *Proceedings of the twenty-second International Conference on Machine Learning*, 2005.

[14] C. Meyer, J. Ganascia et J. Zucker : Learning strategies in games by anticipation. Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, (IJCAI'97)*, 1997.

[15] J. Nash : Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36, 1950.

[16] J. Pérolat, B. Piot, M. Geist, B. Scherrer et O. Pietquin : Softened approximate policy iteration for Markov games. Dans *Proceedings of the International Conference on Machine Learning (ICML 2016)*, 2016.

[17] T. E. S. Raghavan et J. A. Filar : Algorithms for stochastic games – a survey. *Zeitschrift für Operations Research*, 35(6), Nov 1991.

[18] D. M. Roijers, P. Vamplew, S. Whiteson et R. Dazeley : A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research (JAIR)*, 48, 2013.

[19] S. M. Ross : Goofspiel - the game of pure strategy. *Journal of Applied Probability*, (8), 1971.

[20] S. Russell et P. Norvig : *Artificial Intelligence : A Modern Approach*. 2010.

[21] A. Saffidine, H. Finnsson et M. Buro : Alpha-Beta pruning for games with simultaneous moves. Dans *Proceedings of the 26th AAAI Conference (**AAAI**)*, Toronto, Canada, juil. 2012.

[22] L. S. Shapley : Stochastic games. *Proceedings of the National Academy of Sciences (PNAS)*, 39, 1953.

[23] L. S. Shapley : Some topics in two person games. *Annals of Mathematical Studies*, 5, 1964.

[24] T. Smith : *Probabilistic Planning for Robotic Exploration*. Thèse de doctorat, The Robotics Institute, Carnegie Mellon University, 2007.

[25] T. Smith et R. Simmons : Point-based POMDP algorithms : Improved analysis and implementation. Dans *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[26] E. Solal : Stochastic games. *Encyclopedia of Database Systems*, 2009.

[27] D. Szer, F. Charpillet et S. Zilberstein : MAA* : A heuristic search algorithm for solving decentralized POMDPs. Dans *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05)*, 2005.

[28] J. von Neumann : Zur Theorie der Gesellschaftsspiele. *Math. Annalen*, 100, 1928.

[29] E. Walraven et M. T. J. Spaan : Accelerated vector pruning for optimal POMDP solvers. Dans *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017.

Open Loop Execution of Tree-Search Algorithms

Erwan Lecarpentier[1] Guillaume Infantes[1] Charles Lesire[1] Emmanuel Rachelson[2]

[1] DTIS (Traitement de l'Information et Systèmes), ONERA - The French Aerospace Lab,
2 avenue Edouard Belin, 31055 Toulouse, France
[2] DISC (Département d'Ingénierie des Systèmes Complexes), ISAE - Supaero,
10 avenue Edouard Belin, 31055 Toulouse, France

[1] {first name}.{last name}@onera.fr, [2] emmanuel.rachelson@isae-supaero.fr

Abstract

In the context of tree-search stochastic planning algorithms where a generative model is available, we consider online planning algorithms building trees in order to recommend an action. We investigate the question of avoiding re-planning in subsequent decision steps by directly using the sub-tree as an action recommender. Firstly, we propose a method for open loop control via a new algorithm taking the decision of re-planning or not at each time step based on an analysis of the statistics of the sub-tree. Secondly, we show that the probability of selecting a suboptimal action at any depth of the tree can be upper bounded and converges towards zero. Moreover, this upper bound decays in a logarithmic way between subsequent depths. This leads to a distinction between node-wise optimality and state-wise optimality. Finally, we empirically demonstrate that our method achieves a compromise between loss of performance and computational gain.

Keywords

Model-based Reinforcement Learning, Tree Search Algorithms, Open Loop Control, Markov Decision Processes.

1 Introduction

Tree-search based algorithms recently encountered a real success at solving sequential, highly combinatorial problems such as the challenging game of Go [Enzenberger et al., 2010, Silver et al., 2016]. Such algorithms use a generative model of the environment to simulate episodes starting from the current state of the agent [Sutton, 1991, Sutton and Barto, 1998]. This allows the exploration of reachable states and actions and results in the construction of an (unbalanced) scenario tree, that aims at identifying promising branches with a limited computational budget. When the computational budget is exhausted, the recommended action at the root node is applied and a new tree is built in the resulting state. This results overall in a closed loop control process.

We are interested in stochastic problems with large state spaces (e.g. continuous) with a short decision time (budget). In this setting, open loop planning algorithms have proven to be successful [Bubeck and Munos, 2010] and even to outperform [Weinstein and Littman, 2012] the standard approaches that consider closed loop policy trees such as UCT [Kocsis and Szepesvári, 2006]. They seek for optimal sequences of actions (plans) rather than optimal policies despite the sub-optimal nature of a plan in stochastic environments. Indeed, computing the latter prevents feedback on the explored states but allows to break the complexity of the state space exploration. Given a tree computed by an open loop planning algorithm, we propose to keep the sub-tree reached by the application of the recommended action and to directly use it as the main tree for the subsequent time step, without re-planning. What motivates this approach is the saving of the computational cost of tree building for subsequent time steps, hence the reduction of number of calls to the simulator. This feature is highly desirable for agents with short decision time, for instance with a high control frequency (e.g. robots torque control) or low computational resources (e.g. satellite systems). In this framework, Perez et al. [2012a] and Heusner [2011] considered keeping the tree in deterministic environments but observed a negative impact as the sub-trees were systematically kept without analysis.

In this paper, we study the impact of using the subsequent sub-trees as main trees for the next action steps without further re-planning. We claim that in lowly-stochastic environments, the reached performance is comparable to algorithms systematically discarding the tree. Our contribution is threefold. (1) We introduce a new algorithm called OLTA (Section 3), performing a systematic analysis of the sub-tree and taking the decision of re-planning or not at each time step. (2) We upper bound the probability of selecting a suboptimal action within a sub-tree, the sense of optimality being defined in an open loop fashion (Section 4). Additionally, we show that this upper bound decays logarithmically with the sub-tree depth. (3) We show in our experiments the benefit of applying such a method both in terms of performance and computational cost saving (Section 5).

The structure of the paper is as follows: first we set the mathematical background in Section 2; then we introduce the OLTA algorithm in Section 3; we perform a theoretical analysis in Section 4, followed by an empirical study in Section 5; finally we conclude in Section 6.

2 Background

2.1 Markov Decision Process

We model the planning problem as a Markov Decision Process (MDP) where an agent sequentially takes actions with the general goal of maximizing the cumulative return fed back by the environment [Puterman, 2014]. We refer to the state space as S and the action space as A. We suppose the number of actions to be finite with $K = |A|$, thus we write $A = \{a_i\}_{i=1}^{K}$. We also consider that the available actions are independent of the state the agent lies in. The state transition function is stochastic and we note $P(s'|s, a)$ the probability of reaching state s' after taking action a in state s. The reward model is denoted by $r(s, a, s')$ and refers to the scalar reward received while performing the transition (s, a, s'). We assume that this reward function is deterministic. Finally, we suppose the horizon of the MDP is infinite and we note $\gamma \in [0, 1[$ the discount factor which represents the importance of the subsequent collected rewards.

2.2 Tree Representation

When a generative model of the MDP is available, it becomes possible to use it within planning algorithms. Tree-search algorithms use this model in order to build a tree of what may possibly occur in the current situation of the agent [Sutton, 1991, Sutton and Barto, 1998, Silver et al., 2008]. In the stochastic setting with potentially infinitely many states, we use a tree structure similar to the one used by Bubeck and Munos [2010]. The tree built at each time step consists in a look-ahead search of the possible outcomes while following some action plan starting from the current state of the agent $s_0 \in S$. Thus, the root node of the tree is labelled by the unique state s_0. The edges correspond to the K available actions, K being the branching factor of the tree. The tree itself conforms to an ensemble of action sequences, or plans, originating from its root node.

We emphasize the fact that this tree structure implies that we search for a state-independent optimal sequence of actions (open loop plan) which is in general sub-optimal compared to a state-dependent policy search. The THTS family of algorithms in particular [Keller and Helmert, 2013] defines trees with chance and decision nodes while our structure does not apply an equality operator on the sampled states. Following Bubeck and Munos [2010], Weinstein and Littman [2012], we argue that closed-loop application of the first action in optimal open loop plans, although theoretically suboptimal, can be competitive with these methods in practice, while being more sample-efficient.

Since the transition model is stochastic, the non-root nodes are not labelled by a unique state. Instead, every such node is associated to a state distribution resulting from the application of the action plan leading to the considered node and starting from s_0. During the exploration, we consider saving all sampled states at each non-root node. A comprehensive illustration of such a tree can be found in Figure 1. This approach extends straightforwardly to Partially Observable Markov Decision Processes (POMDP) [Silver and Veness, 2010].

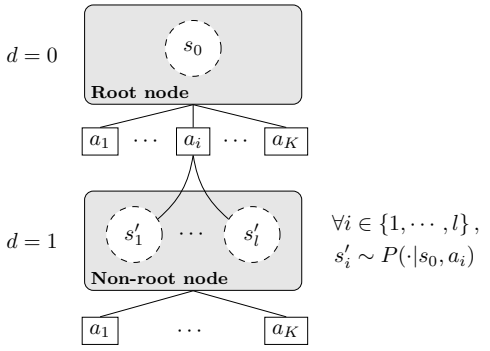


Figure 1: General representation of a tree, where $l \in \mathbb{N}$ is the number of times the sub-tree reached by action a_i has been developed. Two nodes are represented in this tree with their respective depths on the left.

Given a tree-search, open loop planning algorithm, we call $\mathcal{T}_d$ the **tree at depth** $d \in \mathbb{N}$, that is the sub-tree resulting from the application of the d first recommended actions. Hence $\mathcal{T}_0$ denotes the whole tree, $\mathcal{T}_1$ the tree starting from the node reached by the application of the first recommended action and so on.

2.3 Open Loop UCT

For the sake of clarity and in order to clearly separate the tree building properties from the open loop execution presented in the next section, we define an open loop planning algorithm utilizing the presented tree structure that we call Open Loop UCT (OLUCT). The difference between UCT and OLUCT is that OLUCT is not provided with an equality operator over states. Within the THTS terminology, this means that decision and chance nodes do not correspond to a single state but to the state distribution reachable by the action plan leading to the node. Hence decision and chance nodes are associated to the state distribution which makes OLUCT an open loop planning algorithm. The fundamental consequence is that an action value within our tree is computed w.r.t. the parent node's state distribution rather than a single state.

Apart from this, OLUCT uses the same exploration procedure as UCT. Within a node, we note $\overline{X}_{i,u}$ the estimated expected return of action i after u samples of this action. $T_i(t)$ is the number of trials of action i up to time t of the OLUCT procedure. An Upper Confidence Bound (UCB) strategy [Auer et al., 2002] is applied at each node where each action corresponds to an arm of a bandit problem.

The tree policy selects the action I_t with the highest UCB:

$$I_t = \underset{i \in \{1, \cdots, K\}}{\arg\max} \left\{ \overline{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\}, \quad (1)$$

where $c_{t,u}$ is an exploration term ensuring that all actions will be sampled infinitely often (Equation 2).

$$c_{t,u} = 2C_p \sqrt{\frac{ln(t)}{u}} \quad (2)$$

The C_p parameter drives the exploration-exploitation trade-off. The OLUCT tree building procedure is detailed in Algorithm 1.

Algorithm 1: OLUCT tree building procedure

Function createTree(state s):
Parameters: budget n; default policy $\pi_{default}$.
Create root node $\nu_{root}(s)$
for $t \in \{1, \cdots, n\}$ **do**
  ν_{leaf} = Select(ν_{root}); // Select a leaf node w.r.t. the UCT strategy and sample a new state for each encountered node.
  Expand(ν_{leaf}); // Expand the node if not terminal using the generative model.
  Δ = Evaluate($\nu_{leaf}, \pi_{default}$); // Simulate a roll-out using $\pi_{default}$, starting from the last sampled state in ν_{leaf}.
  Backup(ν_{leaf}, Δ); // Back-propagate the sampled return.
return $\mathcal{T}(\nu_{root})$

3 OLTA (Open Loop Tree-search Algorithm)

3.1 Description

In order to control the execution of open loop plans, we propose a new algorithm called OLTA (Algorithm 2). It relies on a generic open loop planning algorithm to generate a tree, rooting from the current state. For the next execution time step, it decides either to use the sub-tree reached by the recommended action or to trigger a re-planning by building a new tree. If no re-planning is triggered, then the recommended action of the sub-tree is applied without using the additional information of the new state observed after the transition. This results in an open loop control process and spares the cost of developing a new tree starting at this state. The intuition behind OLTA is that several consecutive recommended actions in an optimal branch of the tree can be reliable, despite the randomness of the environment. A major example of such a case is low-level control, where consecutive sampled states are close to each other.

In this paper, for its performance and simplicity, we chose to implement OLUCT as the open loop planning algorithm utilized by OLTA. However, any other algorithm generating trees as described in Section 2.2 could be used in the same way (e.g. OLOP [Bubeck and Munos, 2010], or HOLOP [Weinstein and Littman, 2012]).

One important feature of OLTA is the so-called "*decision-Criterion*", based on which the agent decides to either use the first sub-tree following the recommended action, or to re-build a new tree from the current state. The decision is based on a comparison with the characteristics of the resulting sub-tree and the current state of the agent. In the next section, we discuss different decision criteria, leading to the consideration of a family of different algorithms.

Algorithm 2: OLTA algorithm

Function OLTA:
Parameters: initial state s_0; tree building procedure createTree; re-planning criterion decisionCriterion.
$s = s_0$;
$\mathcal{T}$ = createTree(s);
while *s is not terminal* **do**
  if *decisionCriterion*($s, \mathcal{T}$) **then**
    a = recommendedAction($\mathcal{T}$); // Get the first recommended action.
  else
    $\mathcal{T}$ = createTree(s); // Create a new tree from the current state.
    a = recommendedAction($\mathcal{T}$); // Get the first recommended action.
  $\mathcal{T}$ = subTree($\mathcal{T}, a$); // Move the tree to the first sub-tree resulting from the application of a.
  s := realWorldTransitionFunction(s, a);

3.2 Decision Criterion

The simplest implementation of the decision criterion is to keep the sub-tree only if its root node is fully expanded. This means that each action has been sampled at least once. We call the resulting algorithm **Plain OLTA**. It naively trusts the value estimates of the sub-tree, thus applies the whole plan of recommended actions at each depth until it reaches a partially expanded node. Therefore, Plain OLTA is expected to perform better in deterministic environments. In stochastic cases however, those estimates may be biased because of the different sources of uncertainty within the MDP (reward function, state transition function and action selection). For this reason, we seek more robust criteria to base the decision on.

A natural way to decide whether to keep the sub-tree or not is to track if the recommended action is optimal w.r.t. the new state s of the agent. Here we make an important distinction between a **state-wise optimal action** and a **node-wise optimal action**. The first one is the action recommended by the optimal policy in a specific state. We note it $a^* = \arg\max_{a \in A} Q^*(s, a)$, with $Q^* : S \times A \to \mathbb{R}$

the optimal state-action value function. In order to define the second one, we introduce S_d, the state random variable at the root node of $\mathcal{T}_d$. Its distribution results from the application of the d first recommended actions starting from s_0, so $S_d \sim P(\cdot|s_0, a_0, \cdots, a_{d-1}) \equiv P_{S_d}(\cdot)$. The node-wise optimal action maximizes the expected return given the state *distribution* of the node. We note it $a_d^* = \arg\max_{a \in A} Q_d^*(a)$ where $Q_d^* : A \to \mathbb{R}$ is the optimal action value function w.r.t. the state distribution at the root node of $\mathcal{T}_d$, that is $Q_d^*(a) = \mathbb{E}_{s \sim P_{S_d}}(Q^*(s, a))$. Following Bellemare et al. [2017], a distributional Bellman equation can be expressed in terms of three sources of randomness that are: $R : S \times A \to \mathbb{R}$ the stochastic reward function; $X : S \times A \to \mathbb{R}$ the random return; and P^π the transition operator with $P^\pi X(s, a) \overset{D}{:=} X(S', A')$, $S' \sim P(\cdot|s, a)$ and $A' \sim \pi(\cdot|S')$. Mathematically, we have the following distributional Bellman equations:

$$\begin{cases} Q^\pi(s, a) = \mathbb{E}_\pi(X(s, a)) \\ Q_d^\pi(a) = \mathbb{E}_{P_{S_d}}(Q^\pi(s, a)) \end{cases}$$

with $X(s, a) \sim R(s, a) + \gamma P^\pi X(s, a)$ and $S_d \sim P_{S_d}(\cdot)$. Unfortunately, at the root node of $\mathcal{T}_d$ for $d > 0$, open loop tree-search algorithms do not estimate Q^* but Q_d^*. The bias introduced by the state distribution implies that in the general case we have no guarantee that $a^* = a_d^*$. The risk is that the set Ω_{S_d} of possible realizations of S_d can include states where a^* is sub-optimal, in which case the resulting return evaluations would weight in favour of a different action than a^*. In other words — introducing the notion of domination domain for an action a as $\mathcal{D}_a = \{s \in S | \pi^*(s) = a\} \subset S$ — if Ω_{S_d} is not included in $\mathcal{D}_{a^*}$, then the risk of the recommended action to be state-wise sub-optimal is increased. Conversely, if Ω_{S_d} is included in the domination domain of a^*, then the optimal action will be selected given that the budget is "big enough" w.r.t. the chosen tree-search algorithm's performance. Consequently, one should base the decision criterion on the analysis of P_{S_d} and the action domination domains. To compute these domains, Rachelson and Lagoudakis [2010] use the properties of Lipschitz-MDPs. Although the following discussion is inspired by this work, the consideration of Lipschitz-MDPs is out of the scope of this paper. We discuss below the construction of decision criteria that will be illustrated in Section 5.

Current state analysis & POMDP setting. The current state s of the agent can be compared to the empirical state distribution $\overline{P_{S_d}}$ at the root node of the sub-tree. If $P_{S_d}(s)$ is large, then the value estimators are related to the locality of the state space the agent lies in. If not, then the node-wise optimal action may not be state-wise optimal. This consideration supposes to identify a state-metric for which two close states have a high chance to be in the same action domination domain. Alternatively, in the case of a POMDP, a belief distribution on the current state is available instead of the current state itself [Kaelbling et al., 1998]. In such a case, a direct comparison between this distribution and $\overline{P_{S_d}}$ can be performed (e.g. with a Wasserstein metric).

State distribution analysis. The dispersion and multi-modality of $\overline{P_{S_d}}$ could motivate not to re-use a sub-tree. A high dispersion involves the possibility that $\overline{\Omega_{S_d}}$ does not belong to a single action domination domain and a re-planning should be triggered. The same consideration applies in terms of multi-modality. Conversely, a narrow, mono-modal, state distribution is a good hint for Ω_{S_d} to be comprised into a single action domination domain.

Return distribution analysis. A widespread or a multi-modal return distribution for the recommended action in a node may indicate a strong dependency on the region of the state space we lie in. If Ω_{S_d} covers different action domination domains, each of these domains may contribute a different return distribution to the node's return estimates, thus inducing a high variance on this distribution or even a multi-modality. In this case, it could be beneficial to trigger the re-planning. Alternatively, even after re-planning, widespread or multi-modal return distributions can naturally arise as a result of the MDP's reward and transition models.

We do not provide a unique generic method to base the decision criterion on. Indeed, we believe that it is a strongly problem-dependent issue and that efficient heuristics can be built accordingly. However, the analysis of the state and return distributions constitute promising indicators and we exemplify their use in the experiments of the last section.

4 Theoretical Analysis

In this section, we demonstrate that the algorithm asymptotically provides node-wise optimal actions for any sub-tree $\mathcal{T}_d$ of depth d. We first derive an upper bound on the failure probability that converges towards zero when the initial budget n of the algorithm goes to infinity. Then, we characterize the loss of performance guarantees between subsequent depths and show a logarithmic decay of the upper bound. The demonstration unfolds as follows: first we write a lower bound for the number of trials of the actions at the root of $\mathcal{T}_d$ in Lemma 1; then we write an upper bound on the failure probability given a known budget at depth d in Lemma 2; finally we derive a recursive relation between the upper bounds of subsequent trees that leads to our result in Theorem 1.

We note $b(d) \in \mathbb{N}$ the **budget** used to develop $\mathcal{T}_d$ i.e. the number of times the d first recommended actions have been selected by the tree policy. We note $T_{i,t}^d$ the number of times the i^{th} action at the root node of $\mathcal{T}_d$ has been selected by the OLUCT tree policy after t expansions of $\mathcal{T}_d$. Similarly, $\overline{X}_{i,T_{i,t}^d}^d \equiv \overline{X}_{i,t}^d$ denotes the estimate of the return of the i^{th} action at depth d after t expansions of the sub-tree $\mathcal{T}_d$. We write I_t^d the index of the action chosen by the tree policy at depth d after t expansions of $\mathcal{T}_d$. We have:

$$I_t^d = \underset{i \in \{1, \cdots, K\}}{\arg\max} \left\{ \overline{X}_{i,t-1}^d + c_{t-1, T_{i,t-1}^d} \right\}$$

The recommended action at depth d given a budget $b(d)$ is $\widehat{I}^d = \arg\max_{i\in\{1,\cdots,K\}} \overline{X}^d_{i,b(d)}$. Following Kocsis and Szepesvári [2006], we assume that the empirical estimates $\overline{X}^d_{i,t}$ converge and write $X^d_{i,t} = \mathbb{E}\{\overline{X}^d_{i,t}\}$ and $X^d_i = \lim_{t\to\infty} X^d_{i,t}$. Then, we define for $i \in \{1,\cdots,K\} \setminus i^*_d$, $\Delta^d_i = X^d_{i^*_d} - X^d_i$ where we note i^*_d the index of the node-wise optimal action at the root node of $\mathcal{T}_d$. We make the hypothesis that only one action is optimal in a given node. The minimum return difference between a sub-optimal action and the optimal one at depth d is $\delta^d = \min_{i\in\{1,\cdots,K\}\setminus i^*_d}(\Delta^d_i)$.

Lemma 1. ***Lower bound for the number of trials.*** *For any sub-tree $\mathcal{T}_d$ developed with a budget $b(d) > K$, there exist a constant $\rho \geq 0$ such that $T^d_{i,b(d)} \geq \lceil \rho \ln(b(d))\rceil$ for all $i \in \{1,\cdots,K\}$. Furthermore, we have the following sequence of lower bounds for the budget with $\lceil\cdot\rceil$ the ceiling function:*

$$\begin{cases} b(d=0) = n \\ b(d) \geq \lceil \rho \ln(b(d-1))\rceil \end{cases}$$

Proof. The first result is borrowed from Kocsis and Szepesvári [2006] where they show it for a generic bandit problem. The extension to our case with a given budget is straightforward. The sequence of lower bounds can be derived by observing that $b(d) = T^{d-1}_{\widehat{I}^{d-1},b(d-1)}$ and applying the previous lower bound. □

Lemma 2. ***Upper bound on the failure probability at depth d given the budget $b(d)$.*** *For any sub-tree $\mathcal{T}_d$ developed with a budget $b(d) > K$ we have the following upper bound on the failure probability, conditioned by the budget $b(d)$:*

$$P(\widehat{I}^d \neq i^*_d | b(d)) \leq b(d)^{-\frac{\rho}{2}(\delta^d)^2}$$

Proof. Let us first bound the failure probability with the probability of overestimating a suboptimal action and underestimating the optimal one up to $\Delta^d_{\widehat{I}^d}/2$.

$$\begin{aligned}
&P\left(\widehat{I}^d \neq i^*_d \middle| b(d)\right) = P\left(\overline{X}^d_{\widehat{I}^d,b(d)} \geq \overline{X}^d_{i^*_d,b(d)} \middle| b(d)\right) \\
&\leq P\left(\overline{X}^d_{\widehat{I}^d,b(d)} \geq X^d_{\widehat{I}^d,b(d)} + \frac{\Delta^d_{\widehat{I}^d}}{2} \cup \right. \\
&\quad \left. \overline{X}^d_{i^*_d,b(d)} \leq X^d_{i^*_d,b(d)} - \frac{\Delta^d_{\widehat{I}^d}}{2} \middle| b(d)\right) \\
&\leq P\left(\overline{X}^d_{\widehat{I}^d,b(d)} \geq X^d_{\widehat{I}^d,b(d)} + \frac{\Delta^d_{\widehat{I}^d}}{2} \middle| b(d)\right) + \\
&\quad P\left(\overline{X}^d_{i^*_d,b(d)} \leq X^d_{i^*_d,b(d)} - \frac{\Delta^d_{\widehat{I}^d}}{2} \middle| b(d)\right)
\end{aligned}$$

From now on, the proof breaks to the analysis of one of the two terms on the right of the last inequality since both can be considered the same way. Let us consider the first term:

$$\begin{aligned}
&P\left(\overline{X}^d_{\widehat{I}^d,b(d)} \geq X^d_{\widehat{I}^d,b(d)} + \frac{\Delta^d_{\widehat{I}^d}}{2} \middle| b(d)\right) \\
&= \sum_{t=1}^{b(d)} P\left(\overline{X}^d_{\widehat{I}^d,b(d)} \geq X^d_{\widehat{I}^d,b(d)} + \frac{\Delta^d_{\widehat{I}^d}}{2} \middle| T^d_{\widehat{I}^d,b(d)} = t\right) \times \\
&\qquad P\left(T^d_{\widehat{I}^d,b(d)} = t \middle| b(d)\right) \\
&\leq \sum_{t=1}^{b(d)} \exp\left\{-\frac{1}{2}(\Delta^d_{\widehat{I}^d})^2 t\right\} P\left(T^d_{\widehat{I}^d,b(d)} = t \middle| b(d)\right) \\
&\leq \sum_{t=\lceil \rho\ln(b(d))\rceil}^{b(d)} \exp\left\{-\frac{1}{2}(\Delta^d_{\widehat{I}^d})^2 t\right\} P\left(T^d_{\widehat{I}^d,b(d)} = t \middle| b(d)\right) \\
&\leq \exp\left\{-\frac{1}{2}(\Delta^d_{\widehat{I}^d})^2 \lceil \rho\ln(b(d))\rceil\right\} \\
&\leq b(d)^{-\frac{\rho}{2}(\delta^d)^2}
\end{aligned}$$

Where we first write the joint probability, then apply Hoeffding's inequality, followed by Lemma 1 and the fact that a convex combination is upper bounded by its higher element. Similarly to Kocsis and Szepesvári [2006], we shall assume that the UCT constant C_p is appropriately chosen for the tail inequalities to be verified. □

Theorem 1. ***Upper bound on the failure probability at depth d.*** *For an initial budget of n and for any sub-tree $\mathcal{T}_d$ developed with a budget $b(d) > K$, we have the following recursive relation for the upper bound on the failure probability, conditioned by the initial budget n:*

$$P(\widehat{I}^d \neq i^*_d | n) \leq \lceil \rho \ln(b(d-1))\rceil^{-\frac{\rho}{2}(\delta^d)^2}$$

Additionally, for any depth $d \geq 1$ given the initial budget n:

$$\begin{cases} P(\widehat{I}^d \neq i^*_d | n) \leq f^d(n)^{-\frac{\rho}{2}(\delta^d)^2} \\ f : t \mapsto \lceil \rho \ln(t)\rceil \end{cases}$$

Where $f^d = f \circ f^{d-1}$ with $f^1 = f$ and $d > 0$.

Proof. We write the joint probability distribution:

$$\begin{aligned}
P(\widehat{I}^d \neq i^*_d | n) &= \sum_{t=1}^{n} P(\widehat{I}^d \neq i^*_d | n, b(d) = t) P(b(d) = t | n) \\
&\leq \sum_{t=1}^{n} t^{-\frac{\rho}{2}(\delta^d)^2} P(b(d) = t | n) \\
&\leq \sum_{t=\lceil \rho\ln(b(d-1))\rceil}^{n} t^{-\frac{\rho}{2}(\delta^d)^2} P(b(d) = t | n) \\
&\leq \lceil \rho \ln(b(d-1))\rceil^{-\frac{\rho}{2}(\delta^d)^2}
\end{aligned}$$

Where we first applied Lemma 2 and then used the fact that $b(d) = T^{d-1}_{\widehat{I}^{d-1},b(d-1)}$ onto which we apply Lemma 1. Finally, we use the fact that a convex combination is upper bounded by its higher element. The last result comes from the sequence of lower bounds in Lemma 1. □

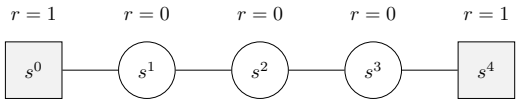


Figure 2: 1D track environment. On top of each cell representing a state is the immediate reward of the transition to this state.

This result shows a logarithmic decay between the upper bounds on the failure probability of two subsequent trees. Asymptotically, at any depth, this upper bound converges towards zero. This result highlights the fact that the deeper the sub-tree is, the less one can rely on the recommended action at the root node. However, we should note that these upper bounds are derived without making further hypotheses on the MDP and express a worst-case value. We show in the next section that equal performances to OLUCT can be reached with a smaller computational budget and number of calls to the generative model.

5 Empirical Analysis

We compared OLUCT with OLTA on a discrete 1D track environment [1] and within a continuous Physical Travelling Salesman Problem [2] (PTSP) [Perez et al., 2012b]. We implemented five decision criteria, leading to five variations of OLTA.

5.1 Heuristic decision criteria

A relevant decision criterion w.r.t. the treated problem allows OLTA to discard a sub-tree when its first recommended action may not be state-wise optimal given the current state of the agent. We implemented five different tests to base this decision on, and evaluated them independently, which led to the following variations of OLTA.

Plain OLTA. The simplest decision criterion that discards a sub-tree only if its root-node is not fully expanded.

State Distribution Modality (SDM-OLTA). Test whether the empirical state distribution is multi-modal or not. If yes, discard the tree if the current state of the agent does not belong to a majority mode. We define a majority mode by a mode comprising more than $\tau_{SDM}\%$ of the sampled states.

State Distribution Variance (SDV-OLTA). Test whether the empirical state distribution variance is above a certain threshold τ_{SDV}. Discard the tree if it is the case. For multi-dimensional state spaces such as in the PTSP, the Variance-Mean-Ratio (VMR) is considered for the different orders of magnitude to be comparable.

State Distance to State Distribution (SDSD-OLTA). Compute the Mahalanobis distance [De Maesschalck et al., 2000] of the current state from the empirical state distribution. Discard the tree if it is above a selected threshold τ_{SDSD}.

Return Distribution Variance (RDV-OLTA). Test whether the empirical return distribution variance is above a certain threshold τ_{RDV}. Discard the tree if it is the case.

A more selective decision criterion can easily be derived by combining the previously described decision criteria and discarding the tree if one of them recommends to do so.

[1] Code available at: https://github.com/erwanlecarpentier/1dtrack.git

[2] Code available at https://github.com/erwanlecarpentier/flatland.git

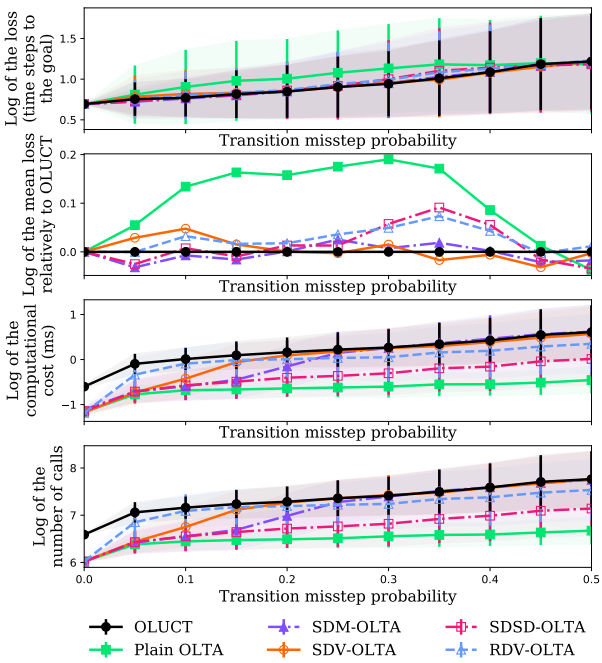
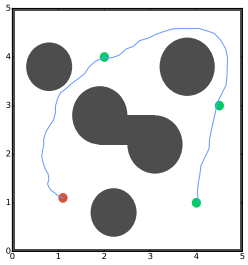


Figure 3: Comparison between OLUCT and OLTA on the discrete 1D track environment for varying values of q.

5.2 1D Track Environment

The 1D track environment (Figure 2), is a 1D discrete world where an agent can either go right or left. The initial state is the "middle" state $s_0 = s^2$. The reward is 0 everywhere except for the transition to the two terminal states s^0 and s^4 for which it is $+1$. The action space is $A = \{right, left\}$. We introduce a transition misstep probability $q \in [0, 1]$ which is the probability to end up in the opposite state after taking an action, for $i \in \{1, 2, 3\}$: $P(s^{i-1}|s^i, right) = q$ and $P(s^{i+1}|s^i, right) = 1 - q$. The same applies for the $left$ action. If $q < 0.5$, the optimal policy $\pi_{optimal}$ is to go left at s^1, to act randomly at s^2 and to go right at s^3. The simulation settings are: $q \in \{0.0, 0.05, \cdots, 0.5\}$; $n = 20$ (budget); $\pi_{default} = \pi_{optimal}$; $H = 10$ (simulation horizon for $\pi_{default}$); $C_p = 0.7$; $\gamma = 0.9$. The decision criteria parameters were tuned to: $\tau_{SDM} = 80$; $\tau_{SDV} = 0.4$; $\tau_{SDSD} = 1$; $\tau_{RDV} = 0.9$. We generated 1000 episodes for each value of q and recorded 3 performance measures: loss (number of time steps to termination); computational cost (measured computation time); and number of calls to the generative model. We display two different graphs of the loss, the second one highlights the relative performance



Trajectory derived by an OLUCT algorithm in our PTSP setting. The starting point is displayed in red, the waypoints in green and the walls in grey.

Figure 4: PTSP illustration

between OLTA and OLUCT.

The motivation behind the use of such a benchmark is to test open loop control in a highly stochastic environment where feedback of the current state is highly informative about the optimal action. In case of misstep for the first action, OLTA has to guess that a re-planning should be triggered while OLUCT does it systematically. As seen on Figure 3, the non-plain OLTA and OLUCT achieved a very comparable loss. Plain-OLTA had a weaker performance due to its systematic re-use of the sub-trees. In terms of both computational cost and number of calls to the generative model, OLTA widely outperforms OLUCT. As q increases, this computational gain vanishes and catches up with OLUCT for SDM-OLTA and SDV-OLTA. This accounts for the discriminative power of their decision criteria that discard more trees. RDV-OLTA and SDSD-OLTA kept a lower computational cost while reasonably matching the performance of OLUCT. Obviously, the computational cost of Plain-OLTA stays low. The apparent similarity between the number of calls to the generative model and the computational cost proves that computing our decision criteria is less expensive than re-planning.

5.3 Physical Travelling Salesman Problem

The PTSP is a continuous navigation problem in which an agent must reach all the waypoints within a maze (Figure 4). The state of the agent is $s = (x, y, \theta, v) \in \mathbb{R}^4$ i.e. the 2D position, orientation and velocity. The action space is $A = \{+d\theta, 0, -d\theta\}$ which consists of the increment, decrement or no-change of the orientation. The reward is $+1$ when a waypoint is reached for the first time, -1 for a wall crash and 0 otherwise. The simulation terminates when the agent reaches all the waypoints or a time limit. The walls cannot be crossed and the orientation is flipped when a crash occurs. We introduce a misstep probability $q \in [0, 1]$ which is the probability for another action to be undertaken instead of the current one. A Gaussian noise of standard deviation σ_{noise} is added to each component of the resulting state from a transition. The simulation settings are: $s_0 = (1.1, 1.1, 0, 0.1)$; $q \in \{0.0, 0.05, \cdots, 0.5\}$; $\sigma_{noise} = 0.02$; $n = 300$ (initial tree budget); $\pi_{default} = \pi_{go-straight}$ that applies no orientation variation; $H = 50$ (simulation horizon for $\pi_{default}$); $C_p = 0.7$; $\gamma = 0.99$. The provided map is the one depicted in Figure 4 with three waypoints. The different decision criteria parameters were tuned to: $\tau_{SDV} = 0.02$; $\tau_{SDSD} = 1$; $\tau_{RDV} = 0.1$. We reserve the development of SDM-OLTA in the continuous case for future work. We generated 100 episodes for each transition misstep probability and recorded the same performance measures as in the 1D track case. The results are presented in Figure 5. OLUCT, SDSD-OLTA and RDV-OLTA achieved a comparable loss for every q, which shows that our method is applicable to larger scale problems than the 1D track environment. SDV-OLTA reached a lower level of performance. Plain OLTA still realized the highest loss since it is highly sensitive to the stochasticity of the environment. In terms of both computational cost and number of calls to the generative model, the same trade-off between performance and computational cost can be observed. Plain OLTA and SDV-OLTA considerably lowered the number of calls at the cost of the performance while SDSD-OLTA and RDV-OLTA realized a better compromise. The number of calls to the generative model and the computational cost are quite similar, meaning that — even with the higher dimensionality of the PTSP compared to the 1D track — the cost incurred by the decision criteria computation is negligible in comparison to the one incurred by the re-planning procedure. Notice that SDV-OLTA achieved a good cost-performance trade-off in the 1D track environment while not in the PTSP relatively to the other algorithms. This is explained by the decision criteria's sensitivity to parameter tuning and by the problem-dependent relevance of such a criterion.

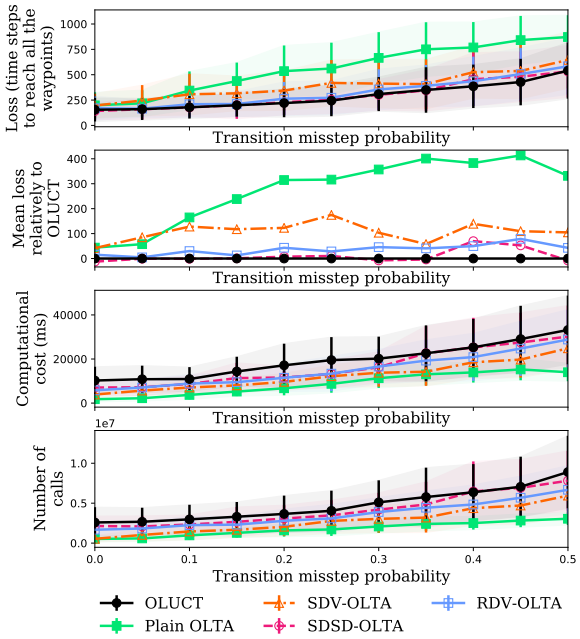


Figure 5: Comparison between OLUCT and OLTA on the continuous PTSP for varying values of q.

6 Conclusion

We introduced OLTA, a new class of tree-search algorithms performing open loop control by re-using subsequent sub-trees of a main tree built with the OLUCT algorithm. A decision criterion based on the analysis of the current state and the current sub-tree allows the agent to efficiently determine if the latter can be exploited. Practically, OLTA can achieve the same level of performance as OLUCT given that the decision criterion is well designed. Furthermore, the computational cost is strongly lowered by decreasing the number of calls to the generative model which is the interest of the approach. We emphasize the fact that this method is generic and can be combined with any other tree-search algorithm than OLUCT. Open questions include building non problem-dependent decision criteria, e.g. by making more restrictive hypothesis on the considered class of MDPs, but also applying the method to other benchmarks and other open loop planners.

References

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

Sébastien Bubeck and Rémi Munos. Open loop optimistic planning. In *COLT*, 2010.

Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

Markus Enzenberger, Martin Muller, Broderick Arneson, and Richard Segal. Fuego - an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

Manuel Heusner. UCT for pac-man. Bachelor thesis, Univ. of Basel, 2011.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon mdps. In *ICAPS*, 2013.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, volume 6, pages 282–293. Springer, 2006.

Diego Perez, Philipp Rohlfshagen, and Simon M Lucas. Monte carlo tree search: Long-term versus short-term planning. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 219–226. IEEE, 2012a.

Diego Perez, Philipp Rohlfshagen, and Simon M Lucas. The physical travelling salesman problem: Wcci 2012 competition. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012b.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Emmanuel Rachelson and Michail G Lagoudakis. On the locality of action domination in sequential decision making. In *ISAIM*, 2010.

David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

David Silver, Richard S Sutton, and Martin Müller. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th international conference on Machine learning*, pages 968–975. ACM, 2008.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

Ari Weinstein and Michael L Littman. Bandit-based planning and learning in continuous-action markov decision processes. In *ICAPS*, 2012.

Génération de scénario : planification avec un opérateur défini par un modèle graphique

R. Lacaze-Labadie[1] D. Lourdeaux[1] M. Sallak[1]

[1] Sorbonne universités, Université de technologie de Compiègne, CNRS,
Heudiasyc UMR 7253, 57 avenue de Landshut – 60203 COMPIEGNE Cedex, France

{remi.lacaze-labadie - domitile.lourdeaux - mohamed.sallak}@hds.utc.fr

Résumé

Utiliser la planification pour générer des scénarios présente de nombreux avantages tels que la variabilité, le contrôle et l'adaptation du scénario. Cependant, un scénario ne doit pas juste être une séquence d'actions valide mais doit également satisfaire la cohérence causale, c'est à dire que la séquence d'actions doit suivre une progression narrative logique et être explicable (i.e. l'apprenant doit comprendre tout ce qui se passe). Dans cet article, nous proposons un nouvel opérateur de planification permettant d'encapsuler des ensembles d'actions sous la forme de graphes. L'intérêt de cet opérateur et de contextualiser les actions et ainsi faciliter la cohérence causale des scénarios générés. Nous donnons un exemple de scénario généré et évaluons les performances de notre système sur un problème des IPCs.

Mots Clef

Planification, Scénarisation, Modèle graphique.

Abstract

Using planning techniques to generate scenarios offer many advantages such as the variability, the control and the adaptation of the scenarios to the learner. However, a scenario is not only a valid sequence of actions/events, it has to satisfy the causal coherence. The causal coherence refers to the notion that the sequence of events has to follow a logical progression of the narrative and the learner has to understand every event occurring in the scenario. In this article, we propose a new operator that allows to encapsulate sets of actions in the form of graphs. This operator allows to contextualize the actions and therefore to facilitate the causal coherence of the generated scenario. We provide an example of a scenario generated using this new operator and we evaluate the performance of our system on IPC problems.

Keywords

Planning, interactive narrative, graphical model.

1 Introduction

Dans cet article nous proposons l'utilisation de techniques de planification pour générer des scénarios dans le cadre de la formation en environnement virtuel (EV). Ce type de formation permet aux apprenants d'expérimenter, de s'entraîner et de voir l'impact de leurs décisions sans risque réel. Nous avons identifié plusieurs contraintes liées à la scénarisation dans ce contexte spécifique qu'est la formation. Premièrement, l'EV doit offrir une grande liberté d'action à l'apprenant [22], tout en maintenant un contrôle scénaristique. Ce contrôle permet à la fois de maintenir l'apprenant autour d'objectifs pédagogiques préalablement choisis par le formateur et également de s'adapter à l'apprenant en faisant varier certains critères de formation tels que la difficulté ou la criticité du scénario. Deuxièmement, le système de scénarisation doit être résilient, c'est à dire être capable de réagir aux actions de l'apprenant qui peut dévier du scénario initialement prévu (i.e "the boundary problem" [15]). Troisièmement, le système doit être capable de proposer un large panel de scénarios afin que l'apprenant puisse répéter la formation, c'est ce que nous appelons la variabilité du système. Enfin, le scénario doit suivre une progression logique, c'est à dire que chaque événement qui le compose doit être explicable.

Génération de scénarios. Pour répondre à ces contraintes, nous devons être capable de générer dynamiquement des scénarios plutôt que de les écrire manuellement. Cela permet de proposer à l'apprenant une grande variabilité de scénarios, alors que des scénarios scriptés seront forcément limités. Cela permet aussi de favoriser la liberté d'action de l'apprenant dans l'EV [20]. En effet, il est possible de générer un nouveau scénario si l'apprenant s'éloigne trop de celui initialement prévu, alors que dans le cas des scénarios scriptés il faudrait prévoir toutes les alternatives possibles, ce qui en pratique est impossible (i.e. "the authoring bottleneck" [21]). Enfin la génération de scénarios offre la possibilité de personnaliser ou d'adapter le scénario en modifiant certaines caractéristiques telles que sa difficulté. La planification s'avère être un bon moyen pour générer des scénarios car

elle formalise les aspects essentiels d'un scénario, à savoir les actions, la temporalité et la causalité [23]. Dans cet article, nous définissons un scénario comme une séquence d'événements qui décrit comment le monde évolue au cours du temps [20, 24, 12]. Par analogie, un plan est une séquence d'actions qui transforme le monde d'un état initial à un état final. Dans cette analogie, les actions au sens de la planification, sont les événements au sens du scénario, c'est pourquoi nous utiliserons ces deux termes de manière interchangeable. Les objectifs scénaristiques sont les buts du problème de planification et les critères de formation (e.g. la difficulté du scénario) sont des coûts attachés aux actions. Nous parlerons donc de coût d'un événement comme un ensemble de niveaux de critères que va engendrer l'apparition de l'événement. Par exemple, *déclencher une coupure de courant* pourra avoir un coût en difficulté de 1 et un coût en stress de 2. Ainsi nous pourrons planifier ce genre d'événements pour générer des scénarios plus ou moins difficiles, ou plus ou moins stressant.

Problématique. Contrairement à des problèmes de planification traditionnels (e.g. logistique, transport...) dans lesquels un plan valide suffit, un scénario doit également satisfaire la cohérence causale. Nous définissons la cohérence causale comme la progression logique des événements qui composent le scénario, dans le but de garantir que tous les événements soient explicables. Un événement est explicable si l'apprenant peut comprendre/expliquer pourquoi il est survenu, ce qui va souvent dépendre du contexte d'apparition de cet événement. En dehors de tout contexte, des événements tels que *faire sonner le téléphone* ou *déclencher une coupure de courant* peuvent ne pas être explicable ou ne pas avoir l'impact escompté. Pour satisfaire la cohérence du scénario, nous devons faire en sorte que ces événements n'arrivent pas par hasard. Si nous prenons l'exemple d'un scénario dans lequel l'apprenant doit s'occuper d'une victime, faire survenir une coupure de courant lorsque l'apprenant n'a aucunement besoin d'électricité n'aura pas le même impact que lorsque celui/celle-ci sera en train de réaliser une opération nécessitant du matériel électrique. L'apprenant peut ne pas comprendre les raisons de cette coupure de courant si cela a peu d'impact dans la réalisation de sa tâche. Il serait possible d'ajouter une précondition spécifiant que la coupure de courant doit se produire seulement si l'apprenant utilise un appareil électrique. Mais limiter cet événement avec une telle précondition ne paraît pas sensé. En effet, la coupure de courant devrait aussi pouvoir survenir dans d'autres circonstances (e.g. simuler un problème de luminosité). Nous avons ici un problème de cohérence causale, dans le sens où la coupure de courant ne sera pas toujours explicable et n'aura pas toujours l'impact escompté. *Rield* montre dans [20], que la cohérence causale ne peut se résoudre simplement en rajoutant des préconditions aux actions. En effet, un des avantages de la planification de scénarios est d'explorer toutes les possibilités en vue de créer des scénarios qui n'étaient même pas envisagés par le formateur. En rajoutant des préconditions pour contextualiser les actions, nous allons grandement limiter l'exploration. Le deuxième problème que nous avons identifié est lié à l'impact de ce type d'événement sur le scénario. Dans la planification classique, le coût d'une action ne dépend que de l'action et de ses variables (e.g. consommation de carburant qui dépend de la distance). A l'inverse, dans le cadre de la scénarisation, les niveaux de critères associés aux événements vont beaucoup varier en fonction du contexte. Par exemple, le coût en difficulté de la coupure de courant sera différent en fonction de ce qu'est en train de faire l'apprenant, chose qu'il n'est pas possible de prendre en compte dans la définition d'une action de planification classique.

Idée générale. Nous cherchons donc un moyen de contextualiser des actions afin de pouvoir évaluer leurs impacts sur le scénario et aussi garantir qu'elles seront planifiées dans un contexte explicable. Pour répondre à cette problématique, nous proposons d'englober des sous-ensembles d'actions dans des graphes, dont les nœuds sont soit des états du monde, soit des actions et dont les arcs sont des relations causales. L'intérêt d'utiliser des graphes d'actions est de pouvoir en extraire des séquences d'actions qui satisfont la cohérence causale. Chaque graphe d'action est défini dans un nouvel opérateur, appelé FRAG (pour fragment de scénario). La planification s'effectue dans l'espace d'actions (traditionnelles) et des FRAGs. Contrairement aux actions, appliquer un FRAG dans le plan consiste d'abord à extraire une séquence d'actions depuis le graphe avant de l'insérer dans le plan. En plus de faciliter la cohérence causale, ce nouvel opérateur permet aussi de définir des coûts contextuels. En effet, une même action peut apparaître dans plusieurs FRAGs (et aussi en tant qu'action classique), ce qui permet de lui associer un coût différent en fonction de son contexte. Par exemple l'événement qui déclenche une coupure de courant pourrait être modélisé dans deux FRAGs. Nous pouvons imaginer un FRAG qui connecte cet événement avec des actions où l'apprenant utilise du matériel électrique et un autre FRAG dans un contexte où l'apprenant réalise des tâches nécessitant une bonne luminosité. Le coût en difficulté de cet événement pourra alors être différent en fonction du contexte.

2 Contexte et travaux connexes

2.1 Cadre d'application

Nos travaux s'inscrivent dans le cadre du projet VICTEAMS (Virtual Characters for Team Training : Emotional, Adaptive, Motivated and Social). Ce projet vise à la création d'un environnement virtuel pour la formation des leaders d'équipe médicale à des compétences non-techniques (e.g. prise de décision, communication, leadership). Le contexte opérationnel est le sauvetage de victimes en temps de guerre ou après une attaque terroriste. Dans cet environnement, l'apprenant joue le rôle du leader médical et les personnages virtuels (PNJs) sont les membres

de l'équipe médicale. Dans la suite, nous utiliserons le terme "formateur" pour désigner la personne qui définit le contenu de la formation et donne les contraintes scénaristiques au planificateur. Nous utiliserons le terme critère de formation ou simplement critère pour désigner un attribut du scénario tel que la difficulté ou la criticité du scénario. Ces critères sont ensuite attachés aux actions et correspondent à l'impact de cette action sur l'apprenant (e.g. faire sonner le téléphone engendre un stress de 2). Notre système fonctionne comme avec des coûts classiques que l'on cherche à minimiser ou maximiser (e.g. le temps, la consommation en énergie). Dans ce projet, le rôle du planificateur est double. Il doit dans un premier temps générer/planifier un scénario qui atteint un but (e.g. sauver toutes les victimes) et satisfait les critères demandés par le formateur. Dans un second temps, le planificateur est en charge de suivre en direct l'exécution du scénario dans l'EV de façon à intervenir (re-planifier) lorsque l'apprenant s'éloigne du scénario initialement prévu. Pour cela, les scénarios ne sont pas composés exclusivement d'événements, mais également d'actions "attendues de l'apprenant" qui sont nécessaire pour le déroulement du scénario. Ce type d'actions fait partie intégrante de la planification, au même titre que les événements, et peut donc être inclu dans des FRAGs. Par exemple, avant l'événement *déclencher une coupure de courant*, on pourra avoir l'action *l'apprenant pose un garrot électronique* dans le plan. Pendant l'exécution du scénario, si une action de l'apprenant est attendue (parce que l'exécution est arrivée sur cette action) mais n'est plus possible (parce que l'action ne satisfait plus ses préconditions dans le monde courant), alors une re-planification est déclenchée.

2.2 Travaux connexes

Nous pouvons classifier les approches de génération de scénarios en deux catégories. Les approches s'appuyant sur la simulation (simulation-based narrative generation) et les approches délibératives (deliberative narrative generation). Dans les approches s'appuyant sur la simulation, le scénario progresse en fonction de l'évolution du monde et des actions prises par les personnages virtuels. C'est par exemple le cas de *Tale-Spin* [16]. On retrouve également dans cette catégorie la narration émergente [2, 1] ou encore les approches centrées sur les personnages [7]. Ces approches considèrent que la narration émerge des interactions entre les personnages virtuels qui évoluent souvent de manière autonome. Ces approches ne permettent pas d'exercer un contrôle externe quant au déroulement du scénario. Le formateur ne pourra donc pas spécifier de contraintes scénaristiques. Les approches délibératives sont celles dont le scénario est créé, souvent via un planificateur, comme une séquence d'actions satisfaisant certaines contraintes et paramètres préalablement choisis. A l'inverse des approches s'appuyant sur la simulation, la génération du scénario est centralisée dans le sens où elle dépend d'un unique auteur ayant le choix de la structure et des contraintes scénaristiques. C'est ensuite au système de trouver un scénario satisfaisant les contraintes demandées. Nous pouvons citer comme exemple le système *Universe* [14] qui utilise un planificateur hiérarchique pour générer des scénarios ou encore les travaux de *Porteous et al.* [18] qui utilisent une variation du planificateur *FF* [10].

Ces deux approches sont aux deux extrémités et la plupart des systèmes sont en réalité hybrides. D'un point de vue scénaristique, nous nous situons vers les approches délibératives qui permettent au formateur de contrôler le scénario. Le formateur joue le rôle de l'auteur en définissant ses objectifs scénaristiques (les buts à atteindre) et ses contraintes (les critères de formation). Toutefois, nous ne générons pas un scénario de A à Z comme on écrirait une histoire. Nous devons plutôt scénariser un monde simulé où l'apprenant est libre de ses actions et les personnages virtuels évoluent en partie de manière autonome. Il faudra alors être capable de re-planifier et adapter le scénario en fonction des divergences entre le scénario prévu et les actions de l'apprenant et des PNJs.

2.3 Limites de la planification

La planification répond aux besoins scénaristiques du projet VICTEAMS. En effet, être capable de planifier des scénarios favorise le contrôle du formateur en lui offrant la possibilité de faire varier des critères de formation (e.g. difficulté du scénario). De plus, il est possible de replanifier depuis n'importe quel état du monde si l'apprenant s'éloigne trop du scénario initialement prévu. Enfin, cela permet de pouvoir proposer une grande viabilité des scénarios sans avoir besoin de tous les imaginer à l'avance. Toutefois, nous avons identifié en introduction des problèmes quant à la cohérence causale des scénarios générés. Prenons un exemple de scénario dans lequel l'apprenant doit gérer l'évacuation de victimes. Dans le cas nominal, il/elle appellera un medevac une fois toutes les victimes examinées, le medevac arrivera et les victimes seront évacuées. Dans notre exemple un medevac est un véhicule aérien servant à évacuer les victimes. Nous souhaitons maintenant augmenter la difficulté de ce scénario. Pour ce faire, le planificateur dispose d'événements perturbateurs dont le coût en difficulté est supérieur à 0, par exemple l'arrivée d'une nouvelle victime. L'ajout d'une victime est une action qui n'a pas de précondition, ce qui fait qu'elle peut être planifiée à tout moment. Toutefois, cela ne sera pas toujours pertinent. Dans notre exemple, la victime peut par exemple être ajoutée avant ou après l'appel au medevac. D'un point de vu formation, il est intéressant de l'ajouter après que le medevac ait été appelé car l'apprenant devra gérer le fait qu'un medevac ne peut pas évacuer plus de victimes que le nombre annoncé lors de son appel. Si par contre la victime arrive avant l'appel au medevac, l'apprenant n'aura pas à gérer cette situation. Il est de plus difficile d'évaluer la difficulté réelle engendrée par cet événement sans en connaître le contexte. Nous proposons donc un moyen de contextualiser ce genre d'événements, à la fois

pour les placer dans des contextes qui permettent d'évaluer leurs impacts sur l'apprenant (en terme de difficulté par exemple) et également pour s'assurer que ces événements apparaissent dans un contexte de formation pertinent. Par exemple, faire arriver une nouvelle victime après que le medevac ait été appelé crée une situation pertinente, du point de vue des formateurs, à laquelle l'apprenant devra faire face. Dans les problèmes de planification traditionnels tels que les problèmes de logistique, la cohérence causal entre les actions et l'explicabilité du plan importe peu, ce qui est recherché c'est un plan valide qui minimise le coût total. Hors, comme le mentionne *Porteous et al.* dans [19], le besoin de générer un plan qui suit une progression logique et raconte une histoire rend la création des domaines de planification bien plus complexe. De plus, *Rield* montre que rajouter des préconditions pour contextualiser les événements limiterait les capacités du planificateur à explorer un grand nombre de possibilités. Nous pensons également qu'ajouter des préconditions et multiplier les actions de planification deviendraient vite maintenable. C'est pourquoi nous proposons un opérateur facilitant la création de ce type de domaine en permettant de modéliser des fragments de scénario via des graphes.

3 Proposition

Nous proposons un nouvel opérateur (appelé FRAG) qui regroupe un ensemble d'actions formant un fragment de scénario, c'est à dire la modélisation d'une situation particulière (e.g. l'arrivée d'une victime alors que l'évacuation a déjà été demandée). L'opérateur FRAG est composé de préconditions et d'une carte cognitive floue (FCM), qui est un graphe de causes/conséquences. Un FRAG est exécuté, puis converti en une séquence d'actions qui est ensuite insérée dans le plan. Ainsi, les actions connectées dans les graphes se retrouvent insérées les unes à la suite des autres, ce qui facilite l'apparition de séquences pertinentes dans le scénario généré. Les FRAGs permettent alors de contextualiser certaines actions, tel que l'ajout d'une victime, en les reliant avec d'autres actions, formant ainsi un contexte situationnel. Une même action peut-être placée dans plusieurs FRAGs et donc dans différents contextes. De plus, cette même action pourra avoir un coût différent pour chaque FRAG, ce qui permet de contextualiser son coût. Enfin, utiliser des graphes plutôt que des séquences d'actions statiques permet d'englober plusieurs alternatives au sein d'un même contexte et de créer des chemins dans le graphe qui engendrerons par exemple plus ou moins de difficulté. La séquence est générée dynamiquement en fonction de l'état courant du monde. Appliquer un FRAG revient donc à (1) initialiser son FCM depuis l'état courant, (2) exécuter le FCM, (3) générer une séquence d'actions depuis le résultat d'exécution du FCM et enfin (4) ajouter cette séquence d'actions dans le plan. Dans la partie suivante, nous donnons plus de détails sur ce qu'est un FCM et sur ces différentes étapes.

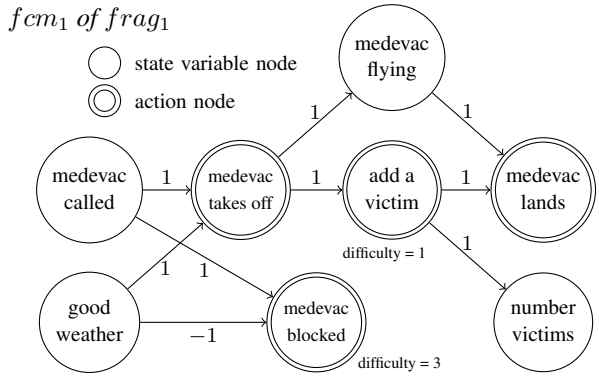


FIGURE 1 – Un exemple simplifié de FCM dans lequel deux alternatives sont possibles en fonction de la météo. L'une des alternatives étant plus difficile que l'autre, le planificateur pourra au préalable changer la météo (via une action classique) pour générer l'alternative correspondante à la difficulté demandée.

3.1 Cartes Cognitives Floues (FCM)

Les cartes cognitives floues, plus connues sous le nom anglais de *Fuzzy Cognitive Map* (FCM), sont des graphes orientés utilisés en théorie de la décision pour modéliser des systèmes complexes de connaissances d'experts. Elles ont été introduites par Kosko en 1986 [13], puis agrémentées de multiples extensions au cours du temps [17]. Un FCM s'appuie sur le principe de concepts (les nœuds du graphe) reliés entre eux par des relations de causes/conséquences (les arcs). Les arcs sont labellisés par un poids, souvent limités à l'intervalle [-1, 1], qui représente le degré d'influence du concept source (la cause) sur le concept destination (la conséquence). Un poids négatif signifie que le concept source refrène/empêche la réalisation du concept destination.

3.2 Exemple de FCM appliqué à la planification

Un exemple de FCM utilisé dans un FRAG est donné en figure 1. Chaque nœud du FCM correspond soit à une action (nœud action), soit à une variable d'état (nœud état). Un arc d'un nœud état vers un nœud action représente une précondition, un arc d'un nœud action vers un nœud état représente un effet et enfin un arc entre deux nœuds actions signifie que l'un doit se produire avant l'autre. Chaque nœud a une fonction d'activation qui régule sa valeur lors de l'exécution du FCM. Exécuter un FCM consiste à itérativement mettre à jour les nœuds du graphe jusqu'à ce que tous les nœuds aient convergé vers leurs valeurs finales. Un nœud a convergé si la différence de sa valeur entre deux itérations est inférieure à un seuil donné. Les nœuds actions sont régulés pour ne prendre que des valeurs de 0 (désactivé) ou 1 (activé). Idem pour les nœuds états associés à des propositions, où 0 signifie que la proposition ne doit pas être présente dans l'état du monde et 1 qu'elle doit l'être. Enfin les nœuds états associés à des variables numériques

peuvent prendre n'importe quelle valeur pourvu que cette valeur appartienne au domaine de la variable numérique. Dans le cadre des FRAGs, les poids des arcs sont limités à 1 sauf pour les arcs liant une proposition à une action qui peuvent aussi avoir un poids de -1. Un arc de -1 signifiant que l'action (nœud destination) peut être activée si la proposition (nœud source) est fausse.
L'exemple que nous proposons sur la figure 1 est un fragment de scénario dans lequel un medevac a été appelé par le leader médical afin d'évacuer les victimes. Par manque de place, les fonctions d'activations ne sont pas visibles sur la figure et les effets ne sont pas tous dessinés. L'idée de ce FCM est de modéliser une situation difficile dans un contexte d'évacuation de victimes. Une fois le medevac appelé par l'apprenant, le FCM propose deux possibilités en fonction de la météo. Si la météo est clémente, une nouvelle victime arrivera après que le medevac ait décollé. La difficulté résidera alors dans le fait qu'un medevac ne peut en théorie embarquer plus de victimes que le nombre établi lors de son appel (car le medevac doit préparer le matériel et le nombre de place en fonction des victimes prévues). Si par contre la météo est mauvaise, la difficulté sera que le medevac ne pourra pas décoller. Nous pouvons voir sur le FCM que dans ce cas, la difficulté estimée par les formateurs est supérieure à la première alternative. Le planificateur pourra prévoir un changement de météo (via une action de planification classique) avant l'appel de ce FRAG, et ainsi planifier l'alternative correspondant à la difficulté demandée. Dans cet exemple, nous pouvons voir que les actions d'ajouter une victime et d'empêcher le medevac de décoller sont contextualisées. De plus, leurs coûts correspondent à la difficulté engendrée dans ce contexte en particulier. Sans FRAG il serait par exemple difficile d'évaluer le réel impact de l'ajout d'une victime sans en connaître le contexte.

4 Formalisation

Dans cette partie, nous décrivons la notation utilisée et formalisons l'opérateur FRAG. Nous nous appuyons sur la notation STRIPS ainsi que sur le langage PDDL2.1 [9] pour les variables numériques.

4.1 Domaine et problème de la planification

Domaine. Un domaine de planification est un système transitoire $\Sigma = (S, A, FRAGS, \gamma, cost)$ où S est un ensemble fini d'états, A est l'ensemble des actions instanciées, FRAGS est l'ensemble des FRAGs instanciés, $\gamma : S \times A$ est la fonction de transition d'un état à un autre, et $cost : A \rightarrow \mathbb{R}_+$ est la fonction coût. Dans notre cas, le coût d'une action est une combinaison linéaire de critères. Un état du monde $s \in S$ est un ensemble de variables d'états $X = P \cup N$ où P est un ensemble de propositions (vrai) et N est un ensemble de variables numériques. Nous utilisons l'hypothèse du monde clos (CWA), ce qui signifie que toutes propositions non présentes dans un état sont considérées comme fausses. Une action A est un tuple $(name_a, pre_a, eff_a)$ où pre_a est l'ensemble des préconditions et eff_a l'ensemble des effets. Ces effets sont eux même subdivisés en un triple $(p^+_{eff}, p^-_{eff}, n_{eff})$ où $p^+_{eff} \subseteq P$ et $p^-_{eff} \subseteq P$ correspondent respectivement aux propositions à ajouter et à supprimer et n_{eff} est l'ensemble des effets numériques sous la forme (n, ass, exp) tel que $ass \in \{:=, +=, -=, *=, /=\}$ et exp représentent une expression. Plus de détails sur les préconditions et effets numériques sont donnés dans [11].

Problème. Un problème de planification est un triplet $P = (\Sigma, s_0, g)$, où Σ est le domaine précédemment défini, s_0 est l'état initial et g est le but du problème sous la forme d'un ensemble de variables d'états. Un état s satisfait g (noté $s \models g$) si toutes les propositions positives de g sont dans s, aucune proposition négative de g sont dans s et toutes les conditions numériques sont satisfaites dans s. Une solution du problème de planification est une séquence d'actions sous la forme $plan = \langle a_1, \ldots, a_n \rangle$ *t.q.* $a_i \in A$. Un scénario est une transformation du plan pour en faire un plan partiellement ordonné $pop = (A_{plan}, \prec_{plan})$ tel que A_{plan} est l'ensemble des actions du plan et $\prec_{plan}$ l'ensemble des contraintes de précédence sous la forme $a_i \prec a_j$ *t.q.* $a \in A$. Cette transformation consiste à ordonner les actions parallèles (qui doivent se produire simultanément) ainsi que les actions sans précondition (événements extérieurs).

Fonction d'évaluation. La fonction d'évaluation est utilisée pour sélectionner le nœud (n) le plus prometteur lors de la planification. Ici, nous ne parlons pas des nœuds d'un FCM mais des nœuds au sens de la planification, c'est à dire représentant l'état du monde. Nous définissons la fonction $f(n) = cost(\pi) + h(s)$ tel que $cost(\pi) = \sum_n^{i=1} cost(a_i)$ est le coût minimal pour atteindre l'état courant et $h : S \rightarrow \mathbb{R}$ est une heuristique indépendante du domaine [4], qui sert à estimer le coût restant pour atteindre un état but (e.g. h^{add} [5], h^{FF} [10]). Traditionnellement le nœud le plus prometteur est celui qui minimise f et nous garderons cette définition dans la partie algorithme pour rester générique. Toutefois, notre cadre d'application nous impose une légère transformation de cette fonction. Nous cherchons un scénario satisfaisant des critères demandés par le formateur. La fonction à minimiser est donc $g(n) = abs(cost\_attendu - f(n))$ de sorte que g est minimal (égale à 0) quand les critères estimés de la solution sont ceux demandés par le formateur.

4.2 L'opérateur FRAG

Un frag est un opérateur de planification défini sous la forme d'une paire (pre, fcm) tel que pre est l'ensemble des préconditions et $fcm \in FCM$, un FCM instancié. Un FCM instancié est un FCM dont les variables d'états associées aux nœuds sont instanciées par les constantes du problème.

Formalisation d'un FCM. Un $fcm \in FCM$ est formalisé comme une paire (C, W) tel que C est un vecteur de

nœuds (les concepts du FCM) de taille n et W est la matrice d'adjacence de taille $n \times n$ qui indique le poids associé à chaque paire de nœuds. On distingue deux types de nœud, à savoir les nœuds actions $C_A \subseteq C$ qui représentent une action exécutable dans l'EV et les nœuds états $C_X \subseteq C$ qui représentent un changement d'état. Parmi ces nœuds états, nous dénotons deux sous-ensembles qui nous servirons dans la partie algorithme : $C_N \subseteq C_X$ pour les nœuds états associés à des variables numériques et $C_P \subseteq C_X$ pour ceux associés à des propositions. Nous définissons donc $cx \in C_X$ et $ca \in C_A$ comme suit :

$$cx = (name, v, f, x) \text{ s.t. } x \in X$$
$$ca = (name, v, f, t, e = \{x_1, \ldots, x_n\}) \text{ s.t. } x_i \in X$$

Les deux types de nœuds ont : un nom, une valeur v et une fonction d'activation f. Chaque nœud état est associé à la variable d'état x qu'il modifie. Après exécution du FCM, une action est générée pour chaque nœud action *activé*, c'est à dire chaque nœud dont la valeur v est égal à 1. Les actions générées ont pour nom le nom du concept, n'ont pas de précondition et ont pour effet des changements d'états associés à la liste de variables d'états e (les détails de la génération des effets sont donnés dans la section algorithme). La variable t est simplement une variable qui contient le nombre d'itération avant que le nœud action ait convergé vers son activation (si il est activé). Cette variable permet de trier par ordre chronologique les nœuds activés de façon à ordonner la séquence d'actions et s'assurer que les causes arrivent avant les conséquences.

Initialisation et exécution d'un FCM. Initialiser un FCM depuis un état courant du monde consiste à initialiser tous ses nœuds actions à 0 (toutes les actions sont désactivées par défaut), puis à initialiser tous ses nœuds états depuis l'état courant (voir section 5). Exécuter un FCM consiste à calculer itérativement la valeur de chaque nœud jusqu'à ce qu'ils aient convergé. Un nœud a convergé quand sa valeur entre deux itérations est en dessous d'un seuil donné. Chaque nœud est mis à jour via sa fonction d'activation (FA) qui calcule la nouvelle valeur du nœud en fonction de la valeur de ses prédécesseurs à l'itération précédente selon l'équation suivante :

$$v_i(t) = f_i(\sum_{\substack{j=1 \\ j \neq i}}^{n} v_j(t-1) w_{ji}) \quad (1)$$

Il existe plusieurs types de fonctions d'activation tels que la fonction linéaire, sinusoïdale ou encore tangente hyperbolique. Le choix de la fonction se fait selon la régulation souhaitée du nœud (e.g. [-1,1], {0,1},...).

5 Algorithme

Notre planificateur utilise un algorithme à chaînage avant, dans lequel la recherche s'effectue en appliquant les actions et les FRAGs. La recherche est guidée par la fonction d'évaluation donnée à la section 4.1. Nous commençons par présenter la fonction $ExecuteFRAG$ qui est en charge d'exécuter un FRAG afin d'en générer une séquence d'actions à insérer dans le plan, puis nous présentons l'algorithme du planificateur.

5.1 Exécution d'un FRAG

Algorithm 1 Exécution d'un FRAG

```
1:  function EXECUTEFRAG(frag, s)
2:      for each c ∈ C_N do
3:          v_c ← x_c(s)
4:      for each c ∈ C_P do
5:          v_c ← 1 if x_c ∈ s else 0
6:      for each c ∈ C_A do
7:          v_c ← 0
8:      execute(fcm_frag)
9:      act_ca = sort({c ∈ C_A | v_c = 1}, key=t)
10:     for each c ∈ act_ca do
11:         // v_cx est la valeur du nœud lié à x (p ou n)
12:         eff ← {p+_eff | p ∈ P ∩ e_c, v_cp = 1} ∪
                  {p-_eff | p ∈ P ∩ e_c, v_cp = 0} ∪
                  {(n_eff, =, v_cn) | n ∈ N ∩ e_c}
13:         a ← (name_c, {}, eff)
14:         seq_a.add(a)
15:     return seq_a
```

La fonction $ExecuteFRAG$ donnée à l'algorithme 1 prend en entrée un FRAG instancié ($frag$) et un état du monde s. La notation fcm_{frag} désigne le FCM de $frag$ et C_N, C_P et C_A désigne les différents types de nœud du FCM (voir section 4). Cette fonction est décomposée en trois étapes : (1) l'initialisation du FCM, (2) l'exécution de ce FCM et enfin (3) la génération de la séquence d'actions en fonction du résultat d'exécution du FCM. L'initialisation du FCM se fait de la ligne 2 à 7 comme suit : tous les nœuds états associés à des variables numériques (C_N) sont initialisés avec la valeur numérique dans s de la variable d'état auquel ils sont associés, tous les nœuds états associés à des propositions (C_P) sont initialisés à 1 si la proposition existe dans s et 0 si elle n'existe pas, et enfin tous les nœuds actions sont inutilisés à 0 (non activé). A la ligne 8, le FCM est exécuté par une librairie externe (e.g. JFCM [8]). La dernière étape consiste à générer une séquence d'actions qui sera insérée dans le plan. Une action est générée pour chaque nœud action activé, c'est à dire qui à une valeur v_c à 1. L'ordre des actions est important, c'est pourquoi la liste des nœuds actions activés est dans un premier temps triée par ordre d'activation pendant l'exécution du FCM (ligne 9). Ensuite, de la ligne 10 à 13, une action est générée pour chaque nœud action activé. Une action générée a le même formalisme que les actions de l'ensemble A. Le nom de l'action correspond au nom du nœud, ces préconditions sont vides et ces effets sont générés depuis la liste de variables d'états e_c. Chaque proposition p dans e_c est ajoutée si la valeur du nœud état associé à p (v_{cp}) est

égale à 1 et retirée de la liste des effets si elle est égale à 0. Nous supposons ici que chaque variable d'état dans e_c a un nœud état qui lui est associé et que sa fonction d'activation produit une valeur de 0 ou 1 (autrement l'effet est ignoré). Chaque variable numérique n dans e_c est ajoutée en tant qu'effet numérique tel que la variable numérique doit être égale à la valeur du nœud état associé a n. Finalement, toutes ces actions sont renvoyées dans une liste.

5.2 Algorithme de planification

Algorithm 2 Algorithme de planification

```
1:  function PLAN(s_0, g, A, FRAGS)
2:      frontier ← {(⟨⟩, s_0)}, explored ← ∅
3:      while frontier ≠ ∅ do
4:          select n ← (π, s) ∈ frontier
5:          frontier.remove(n), explored.add(n)
6:          if s ⊨ g then
7:              return buildPlan(π)
8:          children ← {(π.a, γ(s, a)) | s ⊨ pre_a}
9:          F ← {frag ∈ FRAGS | s ⊨ pre_frag}
10:         for each frag ∈ F do
11:             seq_a ← EXECUTEFRAG(frag, s)
12:             s' ← s
13:             for each a ∈ seq_a do
14:                 s' ← γ(s', a)
15:                 children.add((π.a, s'))
16:         prune(frontier ∪ explored ∪ children)
17:         frontier ← frontier ∪ children
18:     return failure
```

Dans cette partie nous décrivons l'algorithme 2 qui planifie un scénario. A chaque itération, l'algorithme choisi le meilleur état à explorer puis il l'explore en appliquant toutes les actions possibles et en exécutant tous les FRAGs applicables (et en insérant leurs séquences d'actions résultantes).

Détails. On commence par définir $frontier$ comme l'ensemble des nœuds prêt à être explorés et $explored$ comme l'ensemble des nœuds déjà explorés. Un nœud $n \leftarrow (\pi, s)$ est une paire dont π est la politique courant et s l'état courant. Ligne 2, $frontier$ est initialisé avec le nœud initial composé d'une politique vide et de l'état initial s_0 et $explored$ est vide. L'algorithme s'exécute tant que $frontier$ n'est pas vide et qu'aucune solution n'a été trouvée. A chaque itération on sélectionne le nœud le plus prometteur (ligne 4) via $f(n)$. On met ensuite à jour $frontier$ et $explored$ (ligne 5) puis on explore ce nœud sauf s'il satisfait le but du problème g, auquel cas le plan est construit depuis π en remontant toutes les actions de l'état final vers l'état initial. Nous explorons s en appliquant toutes les actions applicables dans s (ligne 8) ainsi que les séquences d'actions générées par l'exécution de tous les FRAGs applicables (ligne 9 à 15). Ligne 16, l'étape d'élagage vise à se débarrasser de tous les nœuds non prometteurs, à savoir tous les nœuds dont l'état s a été atteint par un meilleur nœud. Enfin, $frontier$ est mis à jour avec les nouveaux nœuds atteints (ligne 17) puis ce processus est répété. Il est important de noter qu'un FRAG avec la même initialisation génère toujours la même séquence d'actions. Par conséquent, on réduit le nombre d'appel à $ExecuteFRAG$ en stockant dans un dictionnaire la séquence d'actions générée pour un FRAG et une initialisation donnée, ainsi nous pouvons la retourner directement plutôt que de faire appel plusieurs fois à $ExecuteFRAG$.

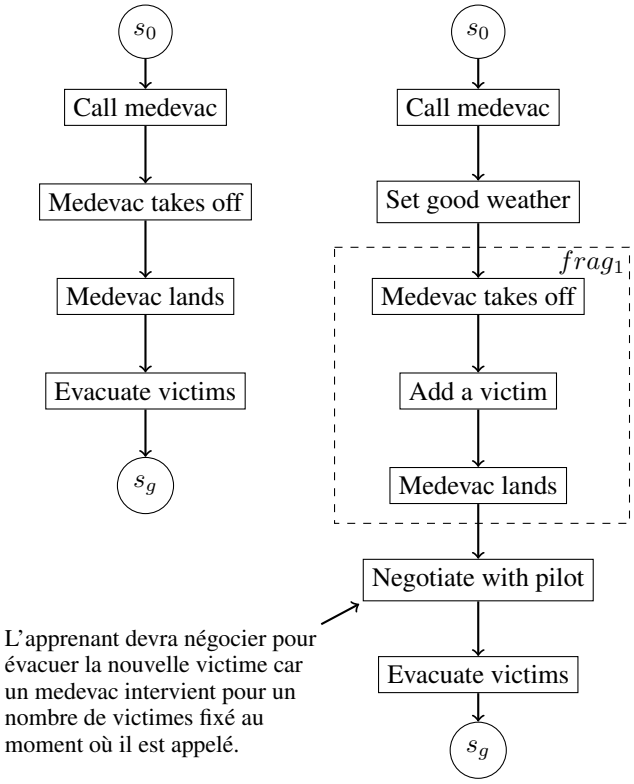


FIGURE 2 – A gauche un scénario de difficulté 0 et à droite un scénario de difficulté 1 qui contient une séquence d'actions provenant du FRAG de la figure 1.

6 Implémentation et exemple

Dans cette partie nous donnons des détails sur l'implémentation de notre planificateur, puis nous présentons un petit exemple de scénario généré. Notre algorithme peut-être facilement implémenté sur la base d'un planificateur faisant de la recherche en chaînage avant. Nous avons choisi $ActuPlan$, un planificateur numérique et temporel développé en Java et proposé par *Beaudry et al.* [3]. Les FCMs sont exécutés avec la librairie JFCM (Java Fuzzy Cognitive Maps) [8]. Et enfin nous utilisons l'heuristique h_{FF} [10].

6.1 Exemple de scénario

Dans l'exemple proposé à la figure 2 le scénario à gauche est le scénario de base (sans aucun critère demandé) dont le but est l'évacuation de victimes. Le scénario généré suppose que l'apprenant va appeler un medevac (medical eva-

cuation), que le medevac va décoller, atterrir sur la zone d'opération, puis les victimes pourront être évacuées. Dans un contexte de formation, nous souhaitons confronter l'apprenant à des situations plus ou moins difficile. Sur le scénario de droite nous avons demandé une difficulté de 1. Le planificateur va utiliser le FRAG défini à la figure 1 et choisir le chemin avec la difficulté de 1. Une météo clémente est d'abord planifiée pour satisfaire la précondition *good weather* du FCM. Une partie du scénario provient donc d'un FRAG qui permet d'ajouter une victime à un moment qui va réellement apporter la difficulté de 1 demandée. Nous pouvons voir un peu plus loin dans le scénario que l'apprenant va devoir négocier avec le pilote pour évacuer cette nouvelle victime avec les autres initialement prévues. Les formateurs ont modélisé que l'ajout d'une victime à ce moment impliquerait de devoir négocier avec le pilote, ce qui engendrerait une difficulté de 1 dans le scénario. L'ajout d'une victime ainsi que son coût en difficulté ont donc pu être contextualisés avec le FRAG de la figure 1.
La capacité de notre système à générer des scénarios cohérents est en cours d'évaluation. Nous prévoyons de générer plusieurs scénarios, avec et sans l'utilisation de FRAGs, et de les proposer à des formateurs pour évaluer leurs pertinences et leurs cohérences. D'un point de vue plus technique, nous évaluons dans la prochaine section les performances de notre planificateur dans le but de déterminer l'impact sur le temps de calcul qu'engendre l'exécution des FRAGs par rapport à la planification classique.

7 Résultats expérimentaux

Dans le but d'avoir un point de comparaison avec des problèmes de planification plus traditionnels, nous avons réalisé nos tests de performance sur des problèmes issus des compétitions internationales de planification (IPC). Nous pensons aussi que cela pourrait ouvrir des perspectives sur l'utilisation de notre opérateur à des problèmes autre que la scénarisation. Ces tests ont été réalisés sur une machine avec un CPU 2.7GHz Intel i7-6820HK et une limite de 4 GB de mémoire pour la JVM. Planifier avec des FRAGs engendre un surcoût car cela implique d'exécuter des FCMs et de générer des séquences d'actions. C'est pourquoi nous évaluons l'impact en terme de performance de leurs utilisations. Nous avons choisi le domaine ZenoTravel [1] qui consiste à embarquer et débarquer des passagers à différents endroits. Ce problème inclus la possibilité de voyager plus ou moins rapidement mais nous considérons dans nos tests une seule vitesse de vol. Les tests s'effectuent sur deux domaines contenant des FRAGs en plus du domaine de base qui lui n'en contient pas. Nous avons comparé le temps d'exécution de ces trois domaines sur des problèmes identiques. Dans un souci de comparaison, les deux domaines définis avec des FRAGs sont tout aussi permissifs que le domaine de base, à savoir que les nœuds actions des FCMs ne contiennent que des actions déjà présentes dans le domaine de base. Le domaine de base, que nous appellerons *no-frag*, a 4 actions qui sont *load*, *fly*, *unload* et *refuel*. Le second domaine appelé *lo-fly-unlo* contient un FRAG qui inclut les actions *load*, *fly* et *unload* et qui peut après exécution générer toutes les séquences possibles de ces 3 actions (en fonction de l'état du monde). Enfin, le troisième domaine appelé *refuel-fly* contient les deux actions *refuel* et *fly*. Les actions non présentes dans les FRAGs sont bien sûr gardées en tant qu'actions classiques.

1. http ://ipc02.icaps-conference.org/

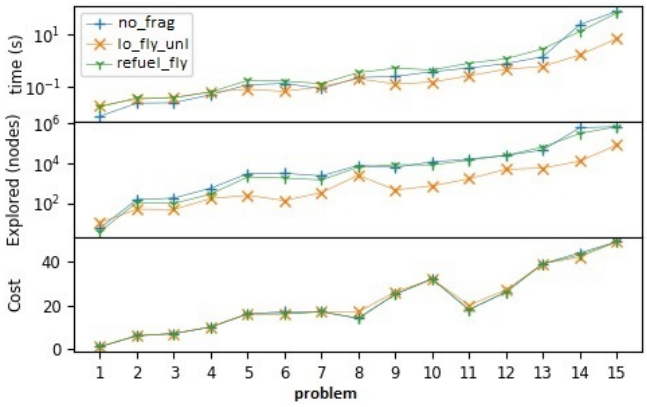


FIGURE 3 – Test de comparaison avec et sans utilisation de FRAGs.

7.1 Analyse des résultats obtenus

La figure 3 présente les résultats que nous avons obtenu sur les 15 premiers problèmes utilisés lors des compétitions internationales de planification (IPCs). Le graphe du bas montre que l'utilisation de FRAGs n'affecte pas le coût total des solutions. Pour se concentrer uniquement sur la comparaison avec et sans FRAGs, le coût de chaque action est de 1 donc le coût total correspond à la taille du plan. Le graphe du haut donne le temps d'exécution (en seconde) et le graphe du milieu donne le nombre de nœuds exploré pendant la recherche (les deux échelles sont logarithmiques). Commençons par comparer le domaine *refuel-fly* et le domaine *no-frag*. Le nombre de nœuds exploré est a peu près identique et le temps d'exécution est légèrement supérieur pour le domaine *refuel-fly*. Nous avons donc bien un temps additionnel d'exécution des FRAGs mais qui ne semble pas significatif. Cela s'explique par le fait que les FCMs s'exécutent assez vite et surtout que le résultat d'exécution est stocké dans un dictionnaire, ce qui réduit grandement le nombre d'exécution de FRAGs. A l'inverse nous pouvons voir que le domaine *lo-fly-unlo* explore moins de nœuds et s'exécute plus rapidement que le domaine *no-frag*. Ceci s'explique par le fait que les actions *load* puis *fly* ou *fly* puis *unload* apparaissent souvent consécutivement dans les solutions. En effet, un avion vole rarement sans au moins charger ou décharger une personne. Ainsi appliquer un FRAG générant la séquence d'actions *{load -> fly}* ou même *{load -> fly -> unload}* fait avancer la recherche vers l'état final plus rapidement. Cet effet n'apparaît pas sur le domaine *refuel-fly* car l'action *refuel*

apparaît finalement peu souvent dans les solutions et donc la séquence *{refuel -> fly}* est peu présente dans les solutions (probablement parce que le réservoir est suffisamment grand pour plusieurs vols). Les séquences d'actions générées se comportent en fait comme des *macro-actions* [6]. Pour résumer, l'exécution de FRAGs engendre effectivement un temps additionnel, qui reste cependant faible. De plus ce temps additionnel peut être compensé par une réduction du nombre d'états explorés si les séquences d'actions générées par les FRAGs apparaissent souvent consécutivement dans les solutions.

8 Conclusion et travaux futurs

Dans cet article, nous avons proposé un nouvel opérateur répondant au besoin de contextualiser des actions et leurs coûts associés. Nous avons vu que cette contextualisation est nécessaire pour générer des scénarios cohérents et qu'il ne suffit pas de rajouter des préconditions aux actions. C'est pourquoi, l'opérateur FRAG permet d'encapsuler un ensemble d'actions reliées entres elles par un graphe. Dans notre cas d'application, un FRAG est représentatif d'un fragment de scénario qui modélise une situation particulière. Par exemple une situation mettant l'apprenant en difficulté, ou une situation qui met l'apprenant face à un choix et mettra donc en jeu des compétences liées à la prise de décision. Ce fragment s'insère dans le scénario sous la forme d'une séquence d'actions. D'un point de vu plus théorique, il est possible d'utiliser les FRAGs pour connecter des actions qui apparaissent souvent dans les solutions et ainsi accélérer la planification. Pouvoir décrire ces situations sous la forme de modèles graphiques aux travers de FCMs permet également une meilleure intégration des connaissances experts. En effet, les FCMs sont des modèles connus pour leurs capacités à modéliser ce type de connaissances. Enfin, l'utilisation de graphes facilite la création des domaines de scénario par les experts, alors que des langages tels que PDDL sont moins abordable pour des non informaticiens.
Nos futurs travaux consistent entre autres à tester plus en profondeur la cohérence des scénarios générés. Nous souhaitons également tester la capacité des formateurs, non familier de la planification, à créer des scénarios à l'aide des FRAGs. Une des limitations de notre système est que les FRAGs génèrent des actions dont le coût ne varie pas en fonction des effets. Nous prévoyons donc de travailler sur la possibilité de définir des coûts variables.

9 Remerciements

Les auteurs remercient la Région Hauts-de-France et le FEDER 2014/2020 pour le cofinancement de ce travail. Celuici a été réalisé dans le cadre du projet VICTEAMS (ANR-14-CE24-0027), financé par l'ANR et la DGA. Il est labellisé par le Labex MS2T.

Références

[1] R. Aylett, S. Louchart, J. Dias, A. Paiva, and M. Vala. FearNot ! - An experiment in emergent narrative. In *Lecture Notes in Computer Science*, volume 3661 LNAI, pages 305–316. 2005.

[2] Ruth Aylett. Narrative in Virtual Environments - Towards Emergent Narrative. In *AAAI fall symposium on narrative intelligence*, pages 83–86, 1999.

[3] Eric Beaudry, Froduald Kabanza, and Francois Michaud. Using a Classical Forward Search to Solve Temporal Planning Problems under Uncertainty. Technical report, 2012.

[4] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2) :5–33, 2001.

[5] Blai Bonet, Gabor Loerincs, and Hector Geffner. A Robust and Fast Action Selection Mechanism for Planning. *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, pages 714–719, 1997.

[6] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-FF : Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24 :581–621, 2005.

[7] Marc Cavazza, Fred Charles, and Steven J. Mead. Character-Based Interactive Storytelling. *IEEE Intelligent Systems*, 17(4) :17–24, 2002.

[8] Dimitri De Franciscis. JFCM : A Java Library for FuzzyCognitive Maps. In Springer, editor, *Fuzzy Cognitive Maps for Applied Sciences and Engineering*, pages 199–220. 2014.

[9] Maria Fox and Derek Long. pddl2.1 : An Extension to pddl for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20 :61–124, 2003.

[10] Jörg Hoffmann. FF : The fast-forward planning system. *AI magazine*, 22 :57–62, 2001.

[11] Jörg Hoffmann. The metric-FF planning system : Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20 :291–341, 2003.

[12] Florence Carre Ineris and Camilla Mörn Msb. Crisis and emergency situation Simulation considering cascading effects methodology. pages 1–28, 2017.

[13] Bart Kosko. Fuzzy cognitive maps. *International Journal of Man-Machine Studies*, 24(1) :65–75, 1986.

[14] Michael Lebowitz. Story-telling as planning and learning. *Poetics*, 14(1985) :483–502, 1985.

[15] Brian S Magerko. Player Modeling in the Interactive Drama Architecture. *Life Sciences*, pages 87–98, 2006.

[16] Jr Meehan. TALE-SPIN, An Interactive Program that Writes Stories. *Ijcai*, pages 91–98, 1977.

[17] Elpiniki I Papageorgiou and Jose L Salmeron. A review of fuzzy cognitive maps research during the last decade. *IEEE Transactions on Fuzzy Systems*, 21(1) :66–79, 2013.

[18] Julie Porteous and Marc Cavazza. Controlling narrative generation with planning trajectories : The role of constraints. *Lecture Notes in Computer Science*, 5915 LNCS :234–245, 2009.

[19] Julie Porteous, Alan Lindsay, Jonathon Read, Mark Truran, and Marc Cavazza. Automated Extension of Narrative Planning Domains with Antonymic Operators. *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, (Aamas) :4–8, 2015.

[20] MO Riedl. Narrative generation : balancing plot and character. *Journal of Artificial Intelligence Research*, 39 :217–268, 2004.

[21] Ulrike Spierling and Nicolas Szilas. Authoring issues beyond tools. *Lecture Notes in Computer Science*, 5915 LNCS :50–61, 2009.

[22] D Thue, V Bulitko, M Spetch, and T Romuanuik. Player agency and the relevance of decisions. In *Interactive Storytelling*, pages 210–150. 2010.

[23] R. Michael Young. Notes on the use of plan structures in the creation of interactive plot. *Narrative Intelligence - Papers from the 1999 AAAI Fall Symposium - TR FS-99-01*, (Walton 1990) :164–167, 1999.

[24] Alexander Zook, Stephen Lee-urban, Mark O Riedl, Heather K Holden, Robert A Sottilare, and Keith W Brawner. Automated Scenario Generation : Toward Tailored And Optimized Military Training In Virtual Environments. *Foundations of Digital Games'12*, page 9, 2012.

Codage SMT dans un espace de plans (liens causaux) pour la planification temporelle en temps continu

Frédéric Maris Maël Valais Julien Vianey
IRIT, Université de Toulouse

{maris, mael.valais, julien.vianey}@irit.fr

Résumé

L'amélioration considérable des solveurs SAT a permis de les utiliser pour la résolution de divers problèmes d'intelligence artificielle, et en particulier la génération automatique de plans d'actions. Récemment, des solveurs SMT (SAT Modulo Theory) prometteurs basés sur des solveurs SAT efficaces ont été développés. Il est donc intéressant d'utiliser le langage SMT qui nous permet de produire directement des codages de problèmes de planification temporelle sans avoir à discrétiser le temps ou à utiliser un gestionnaire de contraintes temporelles externe (Time Map Manager). Cette approche a déjà été explorée, mais la plupart des codages existants considèrent une modélisation discrète du temps. Nous introduisons dans cet article une nouvelle traduction de problèmes de planification temporelle en temps continu, basée sur un codage dans un espace de plans (liens causaux), en formules propositionnelles modulo QF-RDL (Quantifier Free Rational Difference Logic).

Mots Clef

Planification, codages, temps, SAT, SMT.

1 Introduction

Dans le planificateur SATPLAN [8], un problème de planification est transformé en une formule propositionnelle dont les modèles, correspondant aux plans solution, peuvent être trouvés en utilisant un solveur SAT. Cette approche SAT recherche un plan solution de longueur fixe k. En cas d'échec à trouver un tel plan, cette longueur est augmentée avant de recommencer la recherche d'une solution. Cette approche bénéficie directement des améliorations des solveurs SAT [1]. L'exemple le plus évident est le planificateur BLACKBOX [10, 11] (et ses successeurs SATPLAN'04 [7] et SATPLAN'06 [12]). Ces planificateurs ont obtenu la première place dans la catégorie planificateur optimal (en termes de nombre d'étapes du plan) des compétitions internationales de planification [2] IPC-2004 et IPC-2006. C'était inattendu car ces planificateurs étaient essentiellement des mises à jour de BLACKBOX et n'incluaient aucune réelle nouveauté : l'amélioration des performances était principalement due aux progrès du solveur SAT sous-jacent.

De nombreuses améliorations de cette approche originale ont été proposées, notamment via le développement de codages plus compacts et efficaces [9, 3, 13, 14, 18, 19, 20]. Suite à ces travaux, de nombreuses autres techniques similaires pour le codage de problèmes de planification ont été développées : programmation linéaire (LP) [23], problèmes de satisfaction de contraintes (CSP) [2]. En particulier, différents types de codages SMT ont aussi été proposés pour résoudre des problèmes de planification temporelle. Les codages de [22] et [21] se basent sur une représentation discrète du temps, alors que dans [17, 16] nous utilisons des atomes de QF-RDL (Quantifier Free Rational Difference Logic) pour coder un temps continu. Dans la section 2, nous définissons notre modèle de planification temporelle, précédemment introduit dans [1]. Nous décrivons dans la section 3 un nouveau codage SAT basé sur des conditions ouvertes (liens causaux) pour la planification classique. Ensuite, dans la section 4, nous proposons une adaptation SMT de ce codage pour la planification temporelle en temps continu.

2 Planification temporelle

Nous étudions la planification temporelle propositionnelle dans un langage basé sur les aspects temporels de PDDL2.1 [4]. Un fluent est une proposition atomique positive ou négative. Comme dans PDDL2.1, nous considérons que les changements des valeurs des fluents sont instantanés mais que les conditions sur la valeur des fluents peuvent être imposées sur un intervalle. Une action a est un quadruplet $\langle Cond(a), Add(a), Del(a), Constr(a)\rangle$, où l'ensemble des conditions $Cond(a)$ est l'ensemble des fluents qui doivent être vrais pour que a soit exécutée, l'ensemble des ajouts $Add(a)$ est l'ensemble des fluents qui sont établis par a, l'ensemble des retraits $Del(a)$ est l'ensemble des fluents qui sont détruits par a, et l'ensemble de contraintes $Constr(a)$ est un ensemble de

1. http://www.satcompetition.org/
2. http://www.icaps-conference.org/index.php/Main/Competitions

contraintes entre les temps relatifs des événements qui se produisent pendant l'exécution de a. Un événement correspond à l'une des quatre possibilités : l'établissement ou la destruction d'un fluent par une action a, ou le début ou la fin d'un intervalle sur lequel un fluent est requis par une action a. Dans PDDL2.1, les événements ne peuvent se produire qu'au début ou à la fin des actions, mais nous relâchons cette hypothèse de sorte que les événements peuvent se produire à tout moment à condition que les contraintes $Constr(a)$ soient satisfaites. Notons que $Add(a) \cap Del(a)$ peut être non vide. En effet, il n'est pas inhabituel pour une action durative d'établir un fluent au début de l'action et de le détruire à sa fin. Nous pouvons également observer que la durée d'une action, le temps entre le premier et le dernier événement de l'action, n'a pas besoin d'être explicitement stockée.

Nous utilisons la notation $a \rightarrow f$ pour désigner l'événement que l'action a établit le fluent f, $a \rightarrow \neg f$ pour désigner l'événement que a détruit f, et $f \mid\!\rightarrow a$ et $f \rightarrow\!\mid a$, respectivement, pour indiquer le début et la fin de l'intervalle sur lequel a requiert la condition f. Si f est déjà vrai (respectivement, faux) lorsque l'événement $a \rightarrow f$ ($a \rightarrow \neg f$) se produit, nous considérons toujours que a établit (détruit) f. Un plan temporel peut contenir plusieurs instances de la même action, mais par simplicité de notation, nous ferons seulement la distinction entre actions et instances d'action si cela est absolument nécessaire. Nous utilisons la notation $\tau(e)$ pour représenter l'instant dans un plan où un événement e se produit. Pour une action donnée (ou une instance d'action) a, $Events(a)$ représente les différents événements qui constituent sa définition, à savoir $a \rightarrow f$ pour tout f dans $Add(a)$, $a \rightarrow \neg f$ pour tout f dans $Del(a)$, $f \mid\!\rightarrow a$ et $f \rightarrow\!\mid a$ pour tout f dans $Cond(a)$. La définition d'une action a inclut les contraintes $Constr(a)$ sur les instants relatifs aux événements dans $Events(a)$. Comme dans PDDL2.1, nous considérons que la durée entre événements dans $Events(a)$ n'est pas nécessairement fixe et que $Constr(a)$ est un ensemble de contraintes d'intervalle sur des paires d'événements, tels que $\tau(f \rightarrow\!\mid a) - \tau(f \mid\!\rightarrow a) \in [\alpha, \beta]$ pour des constantes α, β. Nous utilisons $[\alpha_a(e_1, e_2), \beta_a(e_1, e_2)]$ pour désigner l'intervalle de valeurs possibles pour la distance relative entre les événements e_1, e_2 de l'action a. Une durée fixe entre les événements $e_1, e_2 \in Events(a)$ peut, bien sûr, être modélisée en définissant $\alpha_a(e_1, e_2) = \beta_a(e_1, e_2)$. De même, l'absence de contrainte peut être modélisée par l'intervalle $[-\infty, +\infty]$. Nous introduisons maintenant deux contraintes fondamentales que tous les plans temporels doivent satisfaire :

— les *contraintes inhérentes* à l'ensemble des instances d'action $\mathcal{A}$: pour toute $a \in \mathcal{A}$, a satisfait $Constr(a)$, c'est-à-dire pour toutes les paires d'événements $e_1, e_2 \in Events(a)$ nous avons $\tau(e_1) - \tau(e_2) \in [\alpha_a(e_1, e_2), \beta_a(e_1, e_2)]$;

— les *contraintes d'effets contradictoires* sur l'ensemble des instances d'action $\mathcal{A}$: pour toutes $a_i, a_j \in \mathcal{A}$, pour tout fluent positif $f \in Del(a_i) \cap Add(a_j)$ nous avons $\tau(a_i \rightarrow \neg f) \neq \tau(a_j \rightarrow f)$.

Les contraintes inhérentes définissent la structure interne de chaque instance d'action, alors que les contraintes d'effets contradictoires assurent que la valeur de vérité de chaque fluent ne soit jamais indéfinie lors de l'exécution d'un plan temporel.

Définition 1 *Un problème de planification temporelle $\langle I, \mathcal{A}, G\rangle$ consiste en un ensemble d'actions $\mathcal{A}$, un état initial I et un but G, où I et G sont des ensembles de fluents.*

Notation Si $\mathcal{A}$ est un ensemble d'instances d'actions, alors $Events(\mathcal{A})$ est l'union des ensembles $Events(a)$ (pour toutes les instances d'action $a \in \mathcal{A}$).

Définition 2 *$P = \langle \mathcal{A}, \tau\rangle$, où $\mathcal{A}$ est un ensemble fini d'instances d'actions $\{a_1, \ldots, a_n\}$ et τ est une fonction à valeurs réelles sur $Events(\mathcal{A})$, est un plan temporel pour le problème $\langle I, \mathcal{A}', G\rangle$ si*

(1) $\mathcal{A} \subseteq \mathcal{A}'$, et

(2) P vérifie les contraintes inhérentes et contradictoires sur $\mathcal{A}$;

et lorsque P est exécuté (c'est-à-dire que les fluents sont établis ou détruits aux instants donnés par τ) à partir de l'état initial I :

(3) pour toute $a_i \in \mathcal{A}$, chaque $f \in Cond(a_i)$ est vrai lorsqu'il est requis, et

(4) tous les fluents $g \in G$ sont vrais à la fin de l'exécution de P.

(5) P est robuste sous des changements infinitésimaux dans les temps de démarrage des actions.

Les événements sont instantanés, alors que les actions ont non seulement une durée mais celle-ci peut aussi être de longueur variable. Ainsi, un plan temporel P ne planifie pas directement ses instances d'action mais planifie tous les événements dans ses instances d'actions. La condition (5) de la définition 2 signifie que nous interdisons les plans qui exigent une synchronisation parfaite entre différentes actions. Cette condition peut être imposée dans PDDL2.1 [5]. Nous exigeons que dans tous les plans, les fluents soient établis strictement avant le début de l'intervalle sur lequel ils sont requis. La seule exception à cette règle est lorsqu'un fluent f est établi et requis par la même action a. Nous permettons la possibilité d'une parfaite synchronisation au sein d'une action, ce qui signifie que nous pouvons avoir $\tau(a \rightarrow f) = \tau(f \mid\!\rightarrow a)$. De même, les fluents ne peuvent être détruits qu'après la fin de l'intervalle sur lequel ils sont requis. La seule exception à cette regle est quand un fluent f est requis

et détruit par une action a, auquel cas on peut avoir $\tau(f \rightarrow| \ a) = \tau(a \rightarrow \neg f)$.

Définition 3 *Un problème de planification temporelle $\langle I, \mathcal{A}, G\rangle$ est positif s'il n'y a pas de fluents négatifs dans les conditions d'actions ni dans le but G.*

Dans cet article, nous ne considèrerons que des problèmes de planification temporelle positifs $\langle I, \mathcal{A}, G\rangle$. Il est bien connu que tout problème de planification peut être transformé en un problème positif équivalent en temps linéaire par l'introduction, pour chaque fluent positif f, d'un nouveau fluent pour remplacer les occurrences de $\neg f$ dans les conditions d'actions [6]. En supposant que tous les problèmes sont positifs, G et $Cond(a)$ (pour toute action a) sont composés de fluents positifs. Par convention, $Add(a)$ et $Del(a)$ sont aussi composés exclusivement de fluents positifs. L'état initial I, cependant, peut contenir des fluents négatifs.

3 Codage SAT pour la planification classique

Dans cette section, nous considérons la planification classique comme restriction de la planification temporelle décrite dans la section précédente. Un problème de planification positif est donc ici un triplet $\langle I, \mathcal{A}, G\rangle$ tel que toutes les actions de $\mathcal{A}$ sont instantanées (c'est-à-dire $\forall a \in \mathcal{A}, \forall e_1, e_2 \in Events(a), \tau(e_1) = \tau(e_2)$). De plus, nous considérons pour tout plan $P = \langle \mathcal{A}, \tau\rangle$, que τ est une fonction à valeurs entières de $Events(\mathcal{A})$ dans $N = \{1, \ldots, n\}$ l'ensemble ordonné des index d'étapes de P.
Les codages SAT dans les espaces de plans existants introduits par [15] se réfèrent tous à trois étapes indexées (pas nécessairement consécutives) du plan. Dans notre nouveau codage, nous allons nous référer seulement à deux étapes consécutives. Pour chaque action $a \in \mathcal{A}$ et chaque étape i, nous créons une variable propositionnelle a_i pour déterminer qu'une instance de a doit être planifiée à l'étape i. Pour chaque fluent $f \in \mathcal{F}$ et chaque étape i, nous créons une variable propositionnelle $open_{f,i}$ pour exprimer que f se maintient à l'étape précédente $i-1$ et doit être protégé au moins jusqu'à l'étape i.

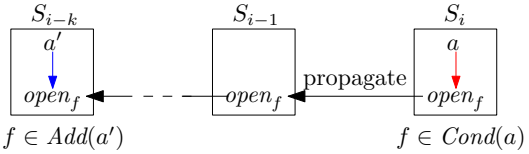


FIGURE 1 – Lien causal : l'action a' produit f à l'étape S_{i-k} pour l'action a qui nécessite f à l'étape S_i.

Dans la figure 1, la variable f est une *condition ouverte* à l'étape S_i, impliquant que $f \in I$ ou une action a' qui ajoute f est exécutée dans une étape précédente S_{i-k}. Les conditions ouvertes doivent être propagées vers l'arrière jusqu'à l'état initial ou une étape dans laquelle elles sont ajoutées par une action.
Dans la suite, nous donnons notre codage pour une longueur de plan fixée *length*. Une borne supérieure pour *length* est le nombre total d'états possibles, soit $2^{|\mathcal{F}|}$.

Conditions ouvertes Si une action a est exécutée dans une étape du plan, alors chaque condition de a doit être une condition ouverte à cette étape (c'est-à-dire qu'un lien causal est requis pour cette condition).

$$\bigwedge_{\mathbf{i}\in[1..length]} \bigwedge_{\mathbf{a}\in\mathcal{A}} \left(\mathbf{a_i} \Rightarrow \bigwedge_{\mathbf{f}\in Cond(\mathbf{a})} open_{\mathbf{f},\mathbf{i}} \right)$$

Dans la dernière étape du plan menant au but tous les fluents du but doivent être des conditions ouvertes ou ajoutées par des actions exécutées dans cette étape.

$$\bigwedge_{\mathbf{f}\in\mathbf{G}} \left(open_{\mathbf{f},length} \vee \bigvee_{\substack{\mathbf{a}\in\mathcal{A} \\ \mathbf{f}\in Add(\mathbf{a})}} \mathbf{a}_{length} \right)$$

Propagation et fermeture Aucune condition ne doit rester ouverte dans la première étape du plan si elle n'est pas fournie dans l'état initial.

$$\bigwedge_{\mathbf{f}\in\mathcal{F}\setminus\mathbf{I}} \neg open_{\mathbf{f},1}$$

Toute condition ouverte dans une étape doit soit rester ouverte soit être ajoutée (fermée) par une action à l'étape précédente.

$$\bigwedge_{\mathbf{i}\in[2..length]} \bigwedge_{\mathbf{f}\in\mathcal{F}} \left(open_{\mathbf{f},\mathbf{i}} \Rightarrow \left(open_{\mathbf{f},\mathbf{i}-1} \vee \bigvee_{\substack{\mathbf{a}\in\mathcal{A} \\ \mathbf{f}\in Add(\mathbf{a})}} \mathbf{a}_{\mathbf{i}-1} \right) \right)$$

Protection des conditions ouvertes Une condition ouverte dans une étape donnée ne peut pas être supprimée à l'étape précédente. Cela garantit de ne rompre aucun lien de causalité dans le plan.

$$\bigwedge_{\mathbf{i}\in[2..length]} \bigwedge_{\mathbf{f}\in\mathcal{F}} \left(open_{\mathbf{f},\mathbf{i}} \Rightarrow \bigwedge_{\substack{\mathbf{a}\in\mathcal{A} \\ \mathbf{f}\in Del(\mathbf{a})}} \neg \mathbf{a}_{\mathbf{i}-1} \right)$$

Prévention des interactions négatives Si une action supprime un fluent qui est nécessaire ou est ajouté par une autre action, alors ces deux actions ne peuvent pas être exécutées toutes les deux dans une même étape.

$$\bigwedge_{\mathbf{i}\in[1..\mathbf{length}]}\ \bigwedge_{\mathbf{a}\in\mathcal{A}}\ \bigwedge_{\mathbf{f}\in(Add(\mathbf{a})\cup Cond(\mathbf{a}))}\ \bigwedge_{\substack{\mathbf{b}\in\mathcal{A}\\ \mathbf{a}\neq\mathbf{b}\wedge\mathbf{f}\in Del(\mathbf{b})}} (\neg\mathbf{a_i}\vee\neg\mathbf{b_i})$$

Dans la section suivante, nous étendons ce codage SAT pour la planification classique en un codage SMT pour la planification temporelle en temps continu.

4 Codage SMT pour la planification temporelle

Nous introduisons une adaptation SMT du codage SAT basé sur les conditions ouvertes que nous avons introduit dans la section précédente pour la planification classique. Ici, les actions ne sont plus instantanées et des contraintes sur les instants $\tau(e)$ auxquels se produisent des événements $e \in Events(\mathcal{A})$ doivent explicitement être ajoutées.
Considérons donc maintenant un problème de planification temporelle positif $\langle I, \mathcal{A}, G\rangle$. Pour chaque instance d'action a et chaque fluent $f \in Cond(a)$, nous introduisons deux variables $\tau_s(open_f)$ et $\tau_e(open_f)$ qui nous permettent de définir un intervalle temporel de protection de lien causal depuis l'état initial ou une étape contenant une action qui produit f vers une étape qui contient l'instance d'action a. Durant cet intervalle temporel, aucune action ne pourra détruire f.

Conditions ouvertes Si une action a est exécutée dans une étape du plan, alors chaque condition de a doit être une condition ouverte à cette étape (c'est-à-dire qu'un lien causal est requis pour cette condition). De plus, le début de l'intervalle sur lequel cette condition est requise se trouve dans l'intervalle de protection du lien causal $[\tau_s(open_f); \tau_e(open_f)]$.

$$\bigwedge_{\mathbf{i}\in[1..\mathbf{length}]}\ \bigwedge_{\mathbf{a}\in\mathcal{A}} \left(\mathbf{a_i} \Rightarrow \bigwedge_{\mathbf{f}\in Cond(\mathbf{a})} \begin{pmatrix} open_{\mathbf{f},\mathbf{i}} \\ \wedge\,(\tau(\mathbf{f}\mid\!\rightarrow \mathbf{a_i}) \geq \tau_s(open_{\mathbf{f},\mathbf{i}})) \\ \wedge\,(\tau(\mathbf{f}\mid\!\rightarrow \mathbf{a_i}) \leq \tau_e(open_{\mathbf{f},\mathbf{i}})) \end{pmatrix}\right)$$

Dans la dernière étape du plan menant au but tous les fluents du but doivent être des conditions ouvertes ou ajoutées par des actions exécutées dans cette étape. Dans le cas où une action a ajoute une condition ouverte f, le début de l'intervalle de protection du lien causal correspondant est fixé à l'instant où a produit f.

$$\bigwedge_{\mathbf{f}\in\mathbf{G}} \left(\begin{array}{l} open_{\mathbf{f},length} \\ \vee \bigvee_{\substack{\mathbf{a}\in\mathcal{A}\\ \mathbf{f}\in Add(\mathbf{a})}} \begin{pmatrix} \mathbf{a}_{length} \\ \wedge\,(\tau(\mathbf{a}_{length}\rightarrow\mathbf{f}) = \tau_s(open_{\mathbf{f},length})) \\ \wedge\,(\tau_{Goal} = \tau_e(open_{\mathbf{f},length})) \end{pmatrix} \end{array}\right)$$

Propagation et fermeture Aucune condition ne doit rester ouverte dans la première étape du plan si elle n'est pas fournie dans l'état initial.

$$\bigwedge_{\mathbf{f}\in\mathcal{F}\setminus\mathbf{I}} \neg open_{\mathbf{f},1}$$

Si une condition ouverte est fournie par l'état initial, alors le début de l'intervalle de protection du lien causal correspondant est fixé à l'instant initial τ_{Init}.

$$\bigwedge_{\mathbf{f}\in\mathbf{I}} (open_{\mathbf{f},1} \Rightarrow (\tau_{Init} = \tau_s(open_{\mathbf{f},1})))$$

Toute condition ouverte f dans une étape doit à l'étape précédente : (1) soit rester ouverte et dans ce cas l'intervalle de protection du lien causal correspondant reste le même pour ces deux étapes, (2) soit être ajoutée (fermée) par une instance d'action a et dans ce cas le début de l'intervalle de protection du lien causal est fixé à l'instant de production de f par a.

$$\bigwedge_{\mathbf{i}\in[2..length]}\ \bigwedge_{\mathbf{f}\in\mathcal{F}} \left(\begin{array}{l} open_{\mathbf{f},\mathbf{i}} \Rightarrow \\ \left(\begin{array}{l} \begin{pmatrix} open_{\mathbf{f},\mathbf{i}-1} \\ \wedge\,(\tau_s(open_{\mathbf{f},\mathbf{i}-1}) = \tau_s(open_{\mathbf{f},\mathbf{i}})) \\ \wedge\,(\tau_e(open_{\mathbf{f},\mathbf{i}-1}) = \tau_e(open_{\mathbf{f},\mathbf{i}})) \end{pmatrix} \\ \vee \bigvee_{\substack{\mathbf{a}\in\mathcal{A}\\ \mathbf{f}\in Add(\mathbf{a})}} (\mathbf{a}_{\mathbf{i}-1} \wedge (\tau(\mathbf{a}_{\mathbf{i}-1}\rightarrow\mathbf{f}) = \tau_s(open_{\mathbf{f},\mathbf{i}}))) \end{array}\right) \end{array}\right)$$

Protection des conditions ouvertes Une condition ouverte dans une étape donnée ne peut pas être supprimée à l'intérieur de l'intervalle de protection du lien causal correspondant $[\tau_s(open_f); \tau_e(open_f)]$. Cela garantit de ne rompre aucun lien de causalité dans le plan.

$$\bigwedge_{\substack{\mathbf{i}\in[1..length]\\ \mathbf{j}\in[1..length]}}\ \bigwedge_{\mathbf{f}\in\mathcal{F}}\ \bigwedge_{\substack{\mathbf{a}\in\mathcal{A}\\ \mathbf{f}\in Del(\mathbf{a})}} \left((open_{\mathbf{f},\mathbf{i}} \wedge \mathbf{a_j}) \Rightarrow \begin{pmatrix} (\tau(\mathbf{a_j}\rightarrow\neg\mathbf{f}) < \tau_s(open_{\mathbf{f},\mathbf{i}})) \\ \vee\,(\tau_e(open_{\mathbf{f},\mathbf{i}}) < \tau(\mathbf{a_j}\rightarrow\neg\mathbf{f})) \end{pmatrix}\right)$$

Prévention des interactions négatives Si une action b supprime, à un instant $\tau(b \rightarrow \neg f)$, un fluent f qui est ajouté par une autre action a à un instant $\tau(a \rightarrow f)$, alors ces deux instants sont différents.

$$\bigwedge_{\substack{\mathbf{i}\in[1..length]\\ \mathbf{j}\in[1..length]}}\ \bigwedge_{\mathbf{a}\in\mathcal{A}}\ \bigwedge_{\mathbf{f}\in Add(\mathbf{a})}\ \bigwedge_{\substack{\mathbf{b}\in\mathcal{A}\\ ((\mathbf{i}\neq\mathbf{j})\vee(\mathbf{a}\neq\mathbf{b}))\wedge\mathbf{f}\in Del(\mathbf{b})}} ((\mathbf{a_i}\wedge\mathbf{b_j}) \Rightarrow (\tau(\mathbf{a_i}\rightarrow\mathbf{f}) \neq \tau(\mathbf{b_j}\rightarrow\neg\mathbf{f})))$$

De même, si une action b supprime, à un instant $\tau(b \rightarrow \neg f)$, un fluent f qui est nécessaire à une autre action a sur un intervalle temporel $[\tau(f \mid\rightarrow a), \tau(f \rightarrow\mid a)]$, alors cet instant ne peut être contenu dans cet intervalle.

$$\bigwedge_{\substack{\mathbf{i}\in[1..length]\\ \mathbf{j}\in[1..length]}} \bigwedge_{\mathbf{a}\in\mathcal{A}} \bigwedge_{\mathbf{f}\in Cond(\mathbf{a})} \bigwedge_{\substack{\mathbf{b}\in\mathcal{A}\\ ((\mathbf{i}\neq\mathbf{j})\vee(\mathbf{a}\neq\mathbf{b}))\wedge\mathbf{f}\in Del(\mathbf{b})}} \left((\mathbf{a_i}\wedge\mathbf{b_j}) \Rightarrow \begin{pmatrix}(\tau(\mathbf{f}\rightarrow\mid \mathbf{a_i}) < \tau(\mathbf{b_j}\rightarrow\neg\mathbf{f}))\\ \vee(\tau(\mathbf{b_j}\rightarrow\neg\mathbf{f}) < \tau(\mathbf{f}\mid\rightarrow \mathbf{a_i}))\end{pmatrix}\right)$$

Bornes du plan temporel Enfin, nous devons ajouter une contrainte pour maintenir le plan dans un intervalle de temps borné par l'étape initiale qui produit l'état initial I et l'étape finale qui nécessite tous les fluents du but G.

$$\bigwedge_{\mathbf{i}\in[1..length]} \bigwedge_{\mathbf{a}\in\mathcal{A}} \left(\mathbf{a_i} \Rightarrow \begin{pmatrix}\bigwedge_{\mathbf{f}\in Cond(\mathbf{a})} \begin{pmatrix}(\tau_{Init}\leq\tau(\mathbf{f}\mid\rightarrow\mathbf{a_i}))\\ \wedge(\tau_{Goal}\geq\tau(\mathbf{f}\rightarrow\mid\mathbf{a_i}))\end{pmatrix}\\ \wedge\bigwedge_{\mathbf{f}\in Add(\mathbf{a})} \begin{pmatrix}(\tau_{Init}\leq\tau(\mathbf{a_i}\rightarrow\mathbf{f}))\\ \wedge(\tau_{Goal}\geq\tau(\mathbf{a_i}\rightarrow\mathbf{f}))\end{pmatrix}\\ \wedge\bigwedge_{\mathbf{f}\in Del(\mathbf{a})} \begin{pmatrix}(\tau_{Init}\leq\tau(\mathbf{a_i}\rightarrow\neg\mathbf{f}))\\ \wedge(\tau_{Goal}\geq\tau(\mathbf{a_i}\rightarrow\neg\mathbf{f}))\end{pmatrix}\end{pmatrix}\right)$$

5 Conclusion et perspectives

Après avoir introduit notre modèle de problèmes de planification temporelle basé sur les aspects temporels de PDDL2.1, nous avons présenté un nouveau codage SAT dans un espace de plans basé sur des conditions ouvertes (liens causaux) pour la planification classique. Nous avons proposé une adaptation SMT de ce codage SAT pour la planification temporelle en temps continu en introduisant des atomes de QF-RDL. Il est maintenant nécessaire de tester ce codage sur les problèmes de référence des compétitions internationales de planification (IPC) et de le comparer avec les autres approches SMT existantes.
Une autre voie à explorer serait d'intégrer cette modélisation des conditions ouvertes dans notre planificateur TLP-GP [17, 16] qui couple un algorithme de recherche dans un graphe de planification temporel à la résolution de contraintes SMT.

Références

[1] Martin C. Cooper, Frederic Maris, and Pierre Régnier. Monotone temporal planning : Tractability, extensions and applications. *J. Artif. Intell. Res. (JAIR)*, 50 :447–485, 2014.

[2] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction : Solving the planning graph by compiling it into CSP. *Artif. Intell.*, 132(2) :151–182, 2001.

[3] M. Ernst, T. Millstein, and D.S. Weld. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*, 1997.

[4] Maria Fox and Derek Long. PDDL2.1 : an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20 :61–124, 2003.

[5] Maria Fox, Derek Long, and Keith Halsey. An investigation into the expressive power of PDDL2.1. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 328–342. IOS Press, 2004.

[6] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[7] H. Kautz. Satplan'04 : Planning as satisfiability. In *Abstracts of the 4th International Planning Competition, IPC-04*, 2004.

[8] H. Kautz and B. Selman. Planning as satisfiability. In *ECAI-92*, pages 359–363, 1992.

[9] H. Kautz and B. Selman. Pushing the envelope : Planning, propositional logic and stochastic search. In *Proc. AAAI-96*, pages 1194–1201, 1996.

[10] H. Kautz and B. Selman. BLACKBOX : A new approach to the application of theorem proving to problem solving. In *Proc. of AIPS-98 Workshop on Planning as Combinatorial Search*, 1998.

[11] H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In *Proc. IJCAI-99*, pages 318–325, 1999.

[12] H. Kautz, B. Selman, and J. Hoffmann. Satplan'06 : Planning as satisfiability. In *Abstracts of the 5th International Planning Competition, IPC-06*, 2006.

[13] A. Mali and S. Kambhampati. Refinement-based planning as satisfiability. In *Proc. Workshop planning as combinatorial search, AIPS-98*, 1998.

[14] A. Mali and S. Kambhampati. On the utility of plan-space (causal) encodings. In *Proc. AAAI-99*, pages 557–563, 1999.

[15] Amol Dattatraya Mali and Subbarao Kambhampati. On the utility of plan-space (causal) encodings. In Jim Hendler and Devika Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pages 557–563. AAAI Press / The MIT Press, 1999.

[16] Frédéric Maris. *Planification SAT et planification temporellement expressive : les systèmes TSP et TLP-GP.* Thèse de doctorat, Université Paul Sabatier, Toulouse, France, 2009.

[17] Frederic Maris and Pierre Régnier. TLP-GP : new results on temporally-expressive planning benchmarks. In *(ICTAI 2008), Vol. 1*, pages 507–514. IEEE Computer Society, 2008.

[18] J. Rintanen. Symmetry reduction for sat representations of transition systems. In *Proc. ICAPS-03*, 2003.

[19] J. Rintanen, K. Heljanko, and Niemelä. Parallel encodings of classical planning as satisfiability. In *Proc. European Conference on Logics in Artificial Intelligence, JELIA-04*, 2004.

[20] J. Rintanen, K. Heljanko, and Niemelä. Planning as satisfiability : parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(1213) :1031–1080, 2006.

[21] Jussi Rintanen. Discretization of temporal models with application to planning with SMT. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3349–3355. AAAI Press, 2015.

[22] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artif. Intell.*, 166(1-2) :194–253, 2005.

[23] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine & its application to resource planning. In Thomas Dean, editor, *Proc. 16th International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, 1999*, pages 310–317. Morgan Kaufmann, 1999.

Re-formation décentralisée d'équipes sous incertitude : modèle et propriétés structurelles

Jonathan Cohen Abdel-Illah Mouaddib

Université de Caen Normandie
prénom.nom@unicaen.fr

Résumé

Notre travail concerne la formation et la re-formation dynamique d'équipes dans le contexte de la planification multi-agent sous incertitude. Nous cherchons à concevoir un système décentralisé qui repose sur les observations individuelles de chacun de ses agents pour ajuster sa composition à l'évolution de la situation. Nous présentons notre modèle ainsi que quelques unes de ses propriétés structurelles intéressantes. Nous montrons également comme notre modèle généralise les définitions fréquemment rencontrées en théorie des jeux coopérative au cas stochastique.

Mots Clef

Planification sous incertitude, systèmes multi-agents, processus décisionnels de Markov, formation d'équipes.

Abstract

This paper describes our work concerning the dynamic formation and reformation of teams in the context of multi-agent planning under uncertainty. We seek to develop a fully decentralized system which uses solely the individual observations of each of its agents in order to adjust its own composition in regards to the evolution of the task at hand. We present our model along with some of its interesting structural properties. We also show how our model generalizes the definitions usually encountered within the realm of cooperative game theory to the stochastic case.

Keywords

Planning under uncertainty, multi-agent systems, Markov decision processes, team formation.

1 Introduction

La formation d'équipe, c'est-à-dire, la capacité à déterminer le nombre et le rôle adéquats des agents au sein d'un groupe d'agents, est une caractéristique essentielle du travail d'équipe. C'est un pré-requis à la bonne réalisation de nombreuses tâches impliquant des systèmes multi-agents. Dans le domaine de la planification multi-agent, où le but est de parvenir à coordonner un groupe d'agents dans l'optique de résoudre une tâche le plus efficacement possible, choisir préalablement une équipe efficace est primordial.

Considérons le cas d'une réponse d'urgence lancée après une catastrophe naturelle. Déterminer à l'avance le nombre de pompiers, d'ambulanciers et de policiers à déployer sur une zone sinistrée va conditionner la réussite de ces agents à minimiser les pertes humaines et matérielles. Une fois l'équipe fixée (généralement par un expert humain), alors il revient à un planificateur (centralisé ou décentralisé) de trouver la stratégie optimale que cette équipe va devoir suivre pour gérer la situation de façon adéquate. Cela est d'autant plus vrai dans les environnements incertains et stochastiques, où les communications entre agents sont limitées voire impossibles, et où ces mêmes agents ne peuvent pas percevoir la totalité de leur environnement, mais uniquement réaliser des observations locales et imprécises.

Les *processus de décision de Markov décentralisés partiellement observables* (Dec-POMDP) sont devenus le cadre mathématique standard *de facto* pour ce qui est du calcul de politiques (quasi-)optimales multi-agents sous incertitude. Malgré la complexité intrinsèque à résoudre efficacement un Dec-POMDP ayant un grand nombre d'agents ou un large espace d'états, les récentes recherches dans le domaine ont permis de développer des algorithmes de résolution efficaces et ayant de bonnes capacités de passage à l'échelle [5, 16, 15, 10].

Bien qu'ils constituent un cadre formel pratique lorsqu'il s'agit de trouver des stratégies efficaces pour des agents autonomes, les Dec-POMDP ne permettent pas d'aborder directement le problème de la re-formation d'équipe. Dans l'intégralité des travaux existants, l'ensemble des agents est donné en entrée, et la planification a lieu en conséquence, en supposant que l'équipe va rester fixe durant toute l'exécution de la tâche. Or, dans beaucoup de cas réels, il peut arriver qu'une défaillance survienne chez un agent et l'empêche de rester actif. Ou bien, l'évolution de la tâche fait que la présence de certains agents se révèle être contre-productive, ou qu'il existe simplement une équipe plus performante. Dans cet article, on se concentre sur ce second aspect, où le processus d'entrée et sortie des agents est complètement contrôlé. On ne va ainsi considérer que les cas où l'entrée/sortie des agents est volontaire et calculée, omettant ainsi les éventuelles pannes et causes extérieures qui pourraient modifier la structure de l'équipe.

Pour en revenir au scénario de la réponse à une situation

de crise, on peut imaginer que, une fois déployés sur place, les agents aient à faire face à des évènements inattendus, ou qui semblaient peu probables avant leur arrivée sur le site, comme l'effondrement soudain d'un bâtiment, ou la propagation plus rapide que prévu d'un feu. Dans ces différents cas, il pourra ainsi être judicieux d'envoyer, ou bien des véhicules de déblaiement, ou bien davantage de pompiers. Selon l'évolution de la situation, l'équipe sera ainsi capable de se re-former pour minimiser à la fois les coûts liés au déploiement d'unités mobiles lourdes ainsi que les pertes liées à la destruction des infrastructures.
La suite de cet article est organisée de la façon suivante. On commence, en partie 2, par rappeler le modèle théorique des Dec-POMDP. Ensuite, en partie 3, nous introduisons notre modèle, le *Team-POMDP*. La partie 4 énumère et analyse plusieurs définitions et propriétés intéressantes, issues de la théorie des jeux coopérative, étendues au cas stochastique et partiellement observable de notre modèle. Nous apportons également une preuve de la complexité théorique de notre modèle. La partie 5 donne quelques pointeurs vers des travaux relatifs aux nôtres et conclue cet article.

2 POMDPs décentralisés

Afin de mieux situer le contexte et comprendre les enjeux scientifiques liés à notre approche, nous commençons par présenter brièvement le modèle sur lequel on se base : le POMDP décentralisé.

2.1 Modèle

Un *processus de décision de Markov décentralisé partiellement observable (Dec-POMDP)* est défini par un tuple

$$\Psi = (\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{O}, p, r, b^0, T)$$

où :

- $\mathcal{N} = \{1, 2, \ldots, n\}$ est l'ensemble fini des agents ;
- $\mathcal{S}$ est l'ensemble fini des états ;
- $\mathcal{A} = \times_{i \in \mathcal{N}} \mathcal{A}_i$ est l'ensemble des actions jointes, et $\mathcal{A}_i$ est l'ensemble des actions individuelles de l'agent i ;
- $\mathcal{O} = \times_{i \in \mathcal{N}} \mathcal{O}_i$ est l'ensemble des observations jointes, et $\mathcal{O}_i$ est l'ensemble des observations individuelles de l'agent i ;
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{O} \to [0, 1]$ est la fonction de dynamique, telle que $p(s, a, s', o) = Pr(s', o \mid s, a)$ est la probabilité de transiter dans l'état s' et d'observer o après avoir effectué l'action jointe a dans l'état s ;
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ est la fonction de récompense, telle que $r(s, a)$ est la récompense reçue lorsque les agents effectuent l'action jointe a dans l'état s ;
- $b^0 \in \Delta S$ est la distribution de probabilité initiale sur les états ;
- et T est l'horizon temporel de planification.

Les Dec-POMDP sont une variation multi-agent et partiellement observable des processus décisionnels de Markov (MDP). Dans un Dec-POMDP, les agents sont autonomes à l'exécution, ils *ne communiquent pas entre eux* et ne partagent aucune information directement (si ce n'est par le biais de leurs interactions avec leur environnement commun). De plus, les agents ne perçoivent que des *observations bruitées* de leur environnement. Cela signifie que ni le point de vue individuel d'un agent, ni la combinaison des points de vue de tous les agents, ne permettent à coup sûr de déterminer l'état réel du système. Au début de l'exécution (avant d'être «lancés» dans le système pour qu'ils réalisent leur tâche), les agents n'ont qu'une connaissance *a priori* de l'état dans lequel le système peut être, connaissance apportée par le paramètre b^0 du Dec-POMDP (qui est donc une hypothèse de départ, apportée par exemple par un expert du domaine. Dans beaucoup de cas étudiés, b^0 va être une distribution uniforme).
Avant de décrire en quoi consiste la résolution d'un Dec-POMDP, il faut introduire les concepts d'*historiques* et de *politiques*.

2.2 Historiques et politiques

L'*historique d'observations individuel* h_i^t d'un agent i à l'instant $t < T$ est la suite des observations reçues par l'agent i jusqu'à l'instant t :

$$h_i^t = (o_i^1, \ldots, o_i^t),$$

avec, pour tout $0 \leq t' \leq t$, $o_i^{t'} \in \mathcal{O}_i$.
L'*historique d'observations joint* à l'instant t est la collection des historiques d'observations individuels de tous les agents :

$$h^t = (h_1^t, h_2^t, \ldots, h_n^t).$$

On note $\mathcal{H}_i^t = \times_t \mathcal{O}_i$ l'ensemble des historiques d'observations individuels de l'agent i à l'instant t, et $\mathcal{H}_i = \cup_t \mathcal{H}_i^t$ l'ensemble de tous les historiques d'observations individuels de l'agent i.
La définition des historiques permet d'introduire la notion essentielle de *politique*. Une *politique individuelle* π_i pour un agent $i \in \mathcal{N}$ est une fonction

$$\pi_i : \mathcal{H}_i \to \mathcal{A}_i.$$

On note Π_i l'ensemble des politiques individuelles de l'agent i. Une *politique jointe* est le tuple correspondant à l'ensemble des politiques individuelles des agents de $\mathcal{N}$:

$$\pi = (\pi_1, \pi_2, \ldots, \pi_n),$$

avec $\pi_i \in \Pi_i$ pour tout agent $i \in \mathcal{N}$.

2.3 Résolution

La *valeur* $V(\pi \mid b_0)$ d'une politique jointe π est la récompense accumulée que l'on peut espérer obtenir en suivant cette politique en supposant une distribution initiale b_0 sur les états :

$$V(\pi \mid b_0) \stackrel{def}{=} \mathbb{E}\left[\sum_{t=0}^{T} r(s^t, a^t)\right],$$

où l'espérance $\mathbb{E}$ est sur les états visités et les observations faites par les agents, $a^t \sim \pi$ est l'action jointe prescrite par la politique jointe à l'instant t, et $r(s^t, a^t)$ est la récompense reçue à l'instant t.
Résoudre un Dec-POMDP, c'est trouver une politique jointe π^*, appelée *politique optimale*, telle que sa valeur soit maximale. Les résultats de complexité liés à la résolution d'un Dec-POMDP ne sont pas optimistes : il a été prouvé que trouver une politique jointe pour un Dec-POMDP (avec plus de 2 agents) ayant une valeur d'au moins K est un problème NEXP-complet [5, 18]. Cela signifie qu'il n'existe pas d'algorithme polynomial pouvant trouver une telle politique, et probablement qu'il n'existe pas non plus d'algorithme exponentiel pouvant la trouver. Cette complexité est due à deux facteurs : l'observabilité partielle et le nombre d'agents : dû à la nature décentralisée du problème, un agent ne peut pas maintenir, durant l'exécution, un état de croyance sur l'état du système. Il doit donc se souvenir de l'ensemble de ses observations. La complexité de la résolution augmente ainsi doublement-exponentiellement avec le nombre d'agents et l'horizon de planification.
Les récentes recherches dans le domaine se sont attachées à trouver des politiques quasi-optimales [13, 20, 11] ou bien à exploiter les propriétés structurelles de certaines sous-classes de Dec-POMDP, telles que l'indépendance des agents ou l'observabilité collective [4, 14]. Les approches optimales ont quant à elles proposé des approches efficaces telles que la compression d'historiques et de politiques [6, 3, 17, 10]. Toutefois, les approches permettant un passage à l'échelle efficace (en terme de nombre d'agents, d'actions, d'observations, d'états et d'horizon de planification) sont encore rares et sujettes à d'actives recherches [22, 10].
Maintenant que nous avons présenté les Dec-POMDP et défini ce que sont les historiques, politiques et valeurs de politiques, nous pouvons introduire notre modèle : le *POMDP d'équipes* (Team-POMDP).

3 POMDP d'équipes

Comme nous l'avons vu, l'une de propriétés intrinsèques d'un Dec-POMDP est l'ensemble des agents $\mathcal{N}$. La planification a lieu dans un système clos : l'équipe dans le système n'a pas vocation à être modifiée durant la totalité du processus de décision. L'ensemble des agents $\mathcal{N}$ reste fixe. Nous présentons quelques approches de travail d'équipe *ad hoc* et de (re-)formation d'équipes dans la section 5. Toutefois, à notre connaissance, la planification dans les systèmes multi-agents dits *ouverts* n'a jamais été étudiée auparavant et constitue donc un domaine de recherche entièrement inexploré.

3.1 Modèle

Afin de faire de la formation et re-formation dans les systèmes multi-agents décentralisés, stochastiques et partiellement observables, nous proposons le modèle de POMDP d'équipe.

Définition 1 (POMDP d'équipe). *Un* processus décisionnel de Markov d'équipes partiellement observable *(Team-POMDP) est défini par le tuple*

$$\Phi = (\mathcal{N}, \mathcal{C}, \mathcal{S}, \{\mathcal{A}_C\}, \{\mathcal{O}_C\}, \{p_C\}, \{r_C\}, b^0, T)$$

où

- $\mathcal{N} = \{1, 2, \ldots, n\}$ *est l'ensemble fini des agents ;*
- $\mathcal{C}$ *est l'ensemble des équipes possibles basées sur la population* $\mathcal{N}$*. Dans le cas le plus général, on prendra* $\mathcal{C} \equiv 2^{\mathcal{N}}$*, l'ensemble des parties de* $\mathcal{N}$ *;*
- $\mathcal{S}$ *est l'ensemble fini des états ;*
- $\mathcal{A}_C = \times_{i \in C} \mathcal{A}_i$ *est l'ensemble des actions jointes de l'équipe* $C, \forall C \in \mathcal{C}$*, et* $\mathcal{A}_i$ *est l'ensemble des actions individuelles de l'agent* $i \in \mathcal{N}$ *;*
- $\mathcal{O}_C = \times_{i \in C} \mathcal{O}_i$ *est l'ensemble des observations jointes de l'équipe* $C \in \mathcal{C}$*, et* $\mathcal{O}_i$ *est l'ensemble des observations individuelles de l'agent* $i \in \mathcal{N}$ *;*
- $p_C : \mathcal{S} \times \mathcal{A}_C \times \mathcal{S} \times \mathcal{O}_C \to [0, 1]$ *est la fonction de dynamique de l'équipe* $C \in \mathcal{C}$*, telle que* $p_C(s, a_C, s', o_C) = Pr(s', o_C \mid s, a_C)$ *est la probabilité de transiter dans l'état* s' *et d'observer* o_C *après que l'équipe* C *ait effectué l'action jointe* a_C *dans l'état* s *;*
- $r_C : \mathcal{S} \times \mathcal{A}_C \times \mathcal{S} \to \mathbb{R}$ *est la fonction de récompense de l'équipe* $C \in \mathcal{C}$*, telle que* $r_C(s, a_C, s')$ *est la récompense reçue lorsque l'équipe* C *effectue l'action jointe* a_C *dans l'état* s *et transite dans l'état* s' *;*
- $b^0 \in \Delta S$ *est la distribution de probabilité initiale sur les états ;*
- *et* T *est l'horizon temporel de planification.*

Remarquons que, dans la définition du tuple, on écrit $\{\mathcal{A}_C\}$ pour désigner le fait que les ensembles d'actions jointes sont définies pour chaque équipe $C \in \mathcal{C}$ (et de façon similaire pour les espaces d'observations jointes, et les fonctions de dynamique et de récompense).
Le modèle de Team-POMDP possède quelques caractéristiques communes avec le modèle des Dec-POMDP. Les principales différences viennent de la nature multiple des différents éléments du tuple : plusieurs fonctions de dynamique, de récompenses, et plusieurs ensembles d'actions et d'observations, un pour chaque équipe possible. Le modèle est une généralisation du modèle standard de Dec-POMDP : un Team-POMDP se ramène à un Dec-POMDP lorsque $|\mathcal{C}| = 1$, c'est-à-dire lorsqu'une seule équipe est autorisée/possible. Comme nous le verrons dans la section suivante, cette observation permet d'affirmer que les jeux d'équipes stochastiques ont une complexité au moins égale à celles des Dec-POMDP.
Un Team-POMDP se déroule principalement de la même façon qu'un Dec-POMDP. L'état initial du système à $t = 0$ est $s^0 \sim b^0$. À chaque étape de décision t, une (et une seule) équipe d'agents $C^t \in \mathcal{C}$ est dite *opérationnelle* dans le système. Cette équipe joue une action jointe a_{C^t} et fait transiter

le système d'un état s^t à un état s^{t+1}, puis l'équipe reçoit l'observation jointe $o^{t+1}_{C^{t+1}}$. À l'étape de décision $t+1$, une nouvelle équipe $C^{t+1} \in \mathcal{C}$ (potentiellement la même qu'à l'instant t), devient l'équipe opérationnelle. [1] C'est elle qui va effectuer une action jointe et recevoir une observation jointe. Ce processus se déroule ainsi jusqu'à ce que l'horizon de planification $t = T$ soit atteint. Dans ce travail, on ne considère que des horizons finis.
Le modèle de Team-POMDP vient avec une difficulté supplémentaire par rapport aux Dec-POMDP. Dans ces derniers, les agents reçoivent des observations à chaque instant t. En maintenant leurs historiques d'observations, ils peuvent utiliser leurs politiques individuelles pour décider quelle action choisir. Or, dans un Team-POMDP, les agents qui ne sont pas dans l'équipe opérationnelle ne perçoivent pas d'observations. C'est la raison pour laquelle nous introduisons les notions d'historiques partiels et de politiques d'équipes.

3.2 Historiques partiels et politiques

L'*historique d'observation partiel individuel* $\dot{h}^t_i$ d'un agent i à l'instant $t < T$ est la suite partielle des observations reçues par l'agent jusqu'à l'instant t :

$$\dot{h}^t_i = (\dot{o}^1_i, \ldots, \dot{o}^t_i)$$

où

$$\dot{o}^{t'}_i = \begin{cases} o^{t'}_i \text{ si } i \in C^{t-1} \\ null \text{ sinon} \end{cases},$$

pour tout $t' \in [1, t]$ et $o^{t'}_i \in \mathcal{O}_i$. La notation $null$ désigne une observation manquante.
L'*historique d'observations partiel joint* à l'instant t est la collection des historiques d'observations partiels individuels des agents présents dans l'équipe opérationnelle $C^t \in \mathcal{C}$:

$$\dot{h}^t = (\dot{h}^t_i \mid i \in C^t).$$

On note $\dot{\mathcal{H}}_i$ l'ensemble des historiques d'observations partiels individuels de l'agent i. Remarquons qu'un historique d'observations partiel se ramène à un historique d'observations individuel classique, complètement spécifié, lorsque toutes les observations sont présentes. Par souci de concision, nous allons simplement parler d'historique individuel $\dot{h}^t_i$ pour désigner l'historique d'observations partiel individuel d'un agent i (et de façon similaire pour les historiques joints).
Une *politique d'équipe individuelle* $\dot{\pi}^i$ pour un agent $i \in \mathcal{N}$ est une fonction

$$\dot{\pi}_i : \dot{\mathcal{H}}_i \to \mathcal{A}_i \cup \{0, 1\}$$

où $\{0, 1\}$ désigne un espace Booléen où 0 correspond à *Ne pas rejoindre l'équipe* et 1 à *Rejoindre l'équipe*. On note $\dot{\Pi}_i$ l'ensemble des politiques d'équipe individuelles de l'agent i.

1. On ne suppose pas de *temps de transition* d'une équipe opérationnelle à l'autre. Par exemple, on néglige le temps de trajet pour sortir de l'équipe ou pour la rejoindre. On suppose que le changement d'équipe opérationnelle se fait entre deux étapes de décisions $t_1, t_2 < T$.

Une *politique d'équipe jointe* $\dot{\pi}$ est le tuple correspondant à l'ensemble des politiques d'équipe individuelles des agents de $\mathcal{N}$:

$$\dot{\pi} = (\dot{\pi}_1, \dot{\pi}_2, \ldots, \dot{\pi}_n),$$

avec $\dot{\pi}_i \in \dot{\Pi}_i$ pour tout agent $i \in \mathcal{N}$.
Comme pour les Dec-POMDP, résoudre un Team-POMDP Φ consiste à trouver une politique d'équipe jointe optimale, notée π^*_Φ, qui va permettre de maximiser la somme espérée des récompenses reçue :

$$\pi^*_\Phi = \underset{\pi_\Phi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{T-1} r_{C^t}(s^t, a^t_{C^t})\right].$$

où l'espérance $\mathbb{E}$ est sur les états et les observations faites par les agents, $a^t_{C^t} \sim \pi^*_\Phi$ est l'action jointe prescrite par la politique d'équipe jointe optimale à l'instant t, et $r_{C^t}(s^t, a^t_{C^t})$ est la récompense reçue à l'instant t.
Dans ce travail, on ne cherche pas de méthode pour résoudre de façon (quasi-)optimale un tel problème. Plutôt, on présente dans la section suivante quelques propriétés intéressantes de notre modèle. La recherche de méthodes et d'algorithmes de résolution seront le sujet de prochains travaux.

4 Propriétés et complexité

Nous commençons dans une première partie par présenter quelques propriétés structurelles de notre modèle. Ces définitions et propriétés sont des concepts habituellement rencontrés dans le domaine de la théorie des jeux coopérative, où les agents doivent former des coalitions afin de maximiser leurs gains [19]. Dans une seconde partie, nous donnons quelques résultats de complexité liés à la résolution des Team-POMDP.

4.1 Propriétés structurelles

On commence par introduire l'opérateur de restriction d'un Team-POMDP à un Dec-POMDP, opérateur qui nous servira plus tard à établir notre principal résultat de complexité.

Définition 2 (Restriction). *Soit Φ un Team-POMDP. Le Dec-POMDP $\Phi|_C$ est la* restriction de Φ *aux agents de l'équipe $C \in \mathcal{C}$, et est défini par :*

$$\Phi|_C = (C, \mathcal{S}, \mathcal{A}_C, \mathcal{O}_C, p_C, r_C, b^0, T).$$

Cette définition permet simplement de formaliser la réduction d'un Team-POMDP à un Dec-POMDP lorsqu'on ne considère qu'une seule équipe opérationnelle possible (c'est-à-dire, pour tout $t < T$, $C^t = C$). En somme, il est possible de voir un Team-POMDP comme une collection de Dec-POMDP qui, lors de l'exécution, alternent les uns avec les autres au gré des décisions transmises par la politique d'équipe jointe.
Nous définissons à présent les propriétés de *monotonie*, de *suradditivité* et de *convexité* des jeux d'équipes stochastiques [9].

Définition 3 (**Monotonie**). *Un Team-POMDP* Φ *est* faiblement monotone *si ses fonctions de récompense* $r_C \in \mathcal{R}$ *sont faiblement monotones, c'est-à-dire, pour toutes équipes* $C, C' \in \mathcal{C}$ *telles que* $C \subseteq C'$*, pour tous états* $s, s' \in \mathcal{S}$ *et pour toutes actions jointes* $a_C \in \mathcal{A}_C, a_{C'} \in \mathcal{A}_{C'}$ *telles que* $a_C \subseteq a_{C'}$*, on a :*

$$r_C(s, a_C, s') \leq r_{C'}(s, a_{C'}, s'). \quad (1)$$

Φ *est* (fortement) monotone *si l'inéquation est stricte.*

Si un Team-POMDP est monotone, cela signifie que les agents n'ont pas d'impact négatif à faire partie de l'équipe opérationnelle. Par exemple, si l'introduction d'agents dans l'équipe opérationnelle comporte un coût positif modélisé d'une certain façon dans les fonctions de récompense du modèle, alors le Team-POMDP sera non-monotone (sauf si la perte liée à l'ajout d'un agent est totalement contrebalancée par le gain que cet agent va apporter à l'équipe).
Un exemple de situation monotone est le scénario de réponse d'urgence décrit en introduction. Imaginons un large feu de forêt qui se répand aux abord d'une zone habitée. L'intuition est que plus le nombre de pompiers mobilisés est grand, meilleure sera la réponse. En effet, comme les pompiers n'interfèrent pas en mal les uns avec les autres et ont un coût de déploiement relativement faible comparé à la valeur qu'ils ajoutent à leur équipe, alors, peu importe le nombre de pompiers déployés, il ne peut être que bénéfique à la mission d'ajouter un autre pompier.

Définition 4 (**Suradditivité**). *Un Team-POMDP* Φ *est* faiblement suradditif *si ses fonctions de récompense* $r_C \in \mathcal{R}$ *sont faiblement suradditives, c'est-à-dire, pour toutes équipes disjointes* $C, C' \in \mathcal{C}$ *telles que* $C \cap C' = \varnothing, C \cup C' \in \mathcal{C}$*, pour tous états* $s, s' \in \mathcal{S}$ *et pour toutes actions jointes* $a_C \in \mathcal{A}_C, a_{C'} \in \mathcal{A}_{C'}$*, on a :*

$$r_C(s, a_C, s') + r_{C'}(s, a_{C'}, s') \leq r_{C \cup C'}(s, (a_C, a_{C'}), s'). \quad (2)$$

Φ *est* (fortement) suradditif *si l'inéquation est stricte.*

Si un Team-POMDP est suradditif, alors il est monotone.
Cette définition spécifie que deux équipes disjointes peuvent être (faiblement) meilleures en se regroupant en une seule équipe. On peut dire d'une certaine manière que le tout est supérieur à la somme des parties. De la définition de suradditivité ci-dessus, on peut dériver la définition suivante de convexité.
Une illustration immédiate montrant l'omniprésence de la notion de suradditivité en situation réelle est notre exemple de catastrophe naturelle. On pourrait envoyer une équipe de pompiers pour empêcher le feu de ravager la zone habitée ; ou bien, on pourrait envoyer une brigade de policiers ériger une périmètre de sécurité et procéder à l'évacuation des civils. Ces deux équipes, si envoyées séparément (l'une après l'autre) pourraient efficacement réussir la mission qui leur incombe, mais elles auraient tout intérêt à coopérer et agir en parallèle pour obtenir de meilleurs résultats.

Définition 5 (**Convexité**). *Un Team-POMDP* Φ *est* convexe *si pour toutes équipes* $C, C' \in \mathcal{C}$ *telles que* $C \subseteq C'$*, pour tous états* $s, s' \in \mathcal{S}$*, pour toutes actions jointes* $a_C \in \mathcal{A}_C, a_{C'} \in \mathcal{A}_{C'}$ *telles que* $a_C \subseteq a_{C'}$*, et pour tout agent* $i \in \mathcal{N}, i \notin C'$*, jouant l'action individuelle* $a_i \in \mathcal{A}_i$ *et tel que* $C \cup \{i\} \in \mathcal{C}$*, on a :*

$$r_{C \cup \{i\}}(s, (a_C, a_i), s') - r_C(s, a_C, s') \leq r_{C' \cup \{i\}}(s, (a'_C, a_i), s') - r_{C'}(s, a'_C, s'). \quad (3)$$

Si un Team-POMDP est convexe, alors il est suradditif.
La propriété de convexité d'un Team-POMDP spécifie que la volonté d'un agent quelconque à rejoindre une équipe augmente proportionnellement avec la taille de cette équipe – Lloyd Shapley parle d'*effet boule de neige* [21].
La propriété de convexité est très forte et rarement présente en pratique. Dans les faits, la plupart des systèmes multi-agents ne sont pas convexes mais plutôt *sous-modulaires*. La sous-modularité est liée à la propriété des rendements décroissants : intuitivement, il s'agit de l'idée selon laquelle ajouter de plus en plus d'agents à l'équipe opérationnelle sera de moins en moins profitable, sans pour autant que cela ne devienne préjudiciable. Notre exemple de feu de forêt est sous-modulaire : à partir d'un certain point, ajouter de nouveaux pompiers n'apportera plus aucune valeur si l'équipe opérationnelle est déjà assez grande.

4.2 Complexité

Nous en venons maintenant aux résultats théoriques de notre travail.

Lemme 1. *Soit* Φ *un Team-POMDP. Si* Φ *est suradditif, alors*

$$\pi^*_\Phi = \pi^*_{\Phi|_\mathcal{N}}.$$

Ce résultat indique que pour résoudre de façon optimale un Team-POMDP suradditif Φ, il est suffisant de trouver la politique optimale du Dec-POMDP défini par la restriction de Φ à la grande équipe $\mathcal{N}$ de tous les agents. C'est un résultat qui suit de la définition de suradditivité, où la grande équipe $\mathcal{N}$ de tous les agents sera toujours au moins aussi performante que n'importe quelle autre équipe.

Théorème 1. *Soit* Φ *un Team-POMDP et* $K \in \mathbb{Z}$ *un nombre entier relatif. Trouver une politique d'équipe jointe pour* Φ *générant un gain d'au moins* K *est un problème NEXP-complet.*

Démonstration. Nous nous servons du fait que le problème de résoudre de façon optimale un Dec-POMDP (avec 2 agents ou plus) est également un problème NEXP-complet [5]. Comme nous l'avons vu, si un Team-POMDP est suradditif, alors il suffit de résoudre le Dec-POMDP issu de la restriction du Team-POMDP à la grande équipe $\mathcal{N}$. Comme un Team-POMDP n'est pas nécessairement suradditif, et comme la restriction est un opérateur simplifiant (puisqu'il permet d'omettre une partie des agents, donc des actions et des observations), alors il suit que résoudre un Team-POMDP est au moins aussi dur que résoudre un Dec-POMDP. □

5 Discussion

Notre résultat de complexité donné en théorème 1 révèle l'intuition que l'on peut avoir lorsqu'on regarde au modèle de Team-POMDP : il semble toujours possible de ramener notre modèle au modèle plus classique des Dec-POMDP. L'idée est de doter les agents d'actions et d'observations fictives nop et nob, qui consistent respectivement à *Ne rien faire* et *Ne rien observer*, et de considérer que la grande équipe $\mathcal{N}$ est l'équipe opérationnelle à chaque étape de décision. De plus, l'espace d'états $\mathcal{S}$ devra également considérer l'état de l'équipe. Une telle approche a déjà été étudiée dans un travail précédent [8]. Bien qu'il soit possible d'effectuer cette transformation, le passage d'un Team-POMDP à un Dec-POMDP voile les propriétés structurelles intéressantes de notre modèle, telles que les définitions décrites dans ce travail. Nous pensons qu'il est possible de se servir de certaines de ces propriétés structurelles pour parvenir à trouver des politiques jointes (quasi-)optimales de façon efficace. De plus, la transformation d'un Team-POMDP vers un Dec-POMDP induit des espaces d'états, d'actions jointes et d'observations jointes bien plus grands, ce qui limite l'applicabilité des méthodes de résolution de Dec-POMDP existantes.

Des travaux similaires aux nôtres ont déjà abordé le problème de (re-)formation d"équipes. Récemment, un article sur les *agents non-dévoués* (*non-dedicated agents*) dans les équipes opérant dans des environnements incertains a abordé une facette du travail décrit ici [1]. Les agents non-dévoués sont des agents susceptibles de quitter l'équipe opérationnelle à un moment donné. Dans [1], l'accent est mis sur le départ d'un agent, et sur la façon dont l'équipe opérationnelle va réagir à un tel départ. L'article propose plusieurs heuristiques, telles que le fait de simplement ignorer le départ de l'agent, ou bien de calculer à nouveau, en réaction au départ, une nouvelle politique aux agents restants. Les différences fondamentales avec notre travail vient du fait que pour eux, un agent i a une probabilité Δ_i^t de quitter l'équipe à un instant t, tandis que nous abordons le problème de la formation et re-formation optimale d'équipes pendant l'exécution, où l'entrée/sortie des agents est calculée et déterministe (relativement aux observations stochastiques faites par les agents). De plus, ils ont une vision centralisée et totalement observable du problème.

Également, on peut citer les méthodes de *travail d'équipe ad hoc* [24, 23, 2]. Dans ce contexte particulier, un agent, appelé l'*agent ad hoc*, rejoint un ensemble d'agents équipiers, auparavant inconnus. L'agent ad hoc n'a pas de coordination préalable et il n'y a pas de communication supposée possible entre lui et les autres agents. Peut-être que ceux-ci ont été développés par une tierce-partie. Contrairement à ce que l'on présente avec notre modèle, la plupart des recherches réalisées dans le domaine du travail d'équipe ad hoc se concentre sur le contrôle d'un seul et unique agent, l'agent ad hoc. De plus, celui-ci ne peut pas quitter son équipe et doit faire face à des coéquipiers potentiellement égoïstes, ou tout du moins qui ne vont pas nécessairement avoir à justifier leurs actions auprès de lui. Des extensions récentes aux cas multi-agents existent [7], mais la nature inhérente du travail d'équipe ad hoc rend de toute façon son étude hors de la portée de nos recherches.

Finalement, il pourrait être intéressant d'étudier les jeux à population incertaine [12] ou encore la planification stochastique décentralisée avec anonymat dans les interactions [25]. Il s'agit de modèles de jeux théoriques où les récompenses reçues par les équipes dépendent du nombre (et du type) des agents, et pas seulement sur leurs stratégies individuelles.

Références

[1] P. Agrawal and P. Varakantham. Proactive and reactive coordination of non-dedicated agent teams operating in uncertain environments. In *IJCAI*, 2017.

[2] S. V. Albrecht and S. Ramamoorthy. A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1155–1156. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[3] R. Aras and A. Dutech. An investigation into mathematical programming for finite horizon decentralized POMDPs. *Journal of Artificial Intelligence Research*, 37 :329–396, 2010.

[4] R. Becker, S. Zilberstein, V. R. Lesser, and C. V. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22 :423–455, 2004.

[5] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research*, 27(4) :819–840, 2002.

[6] A. Boularias and B. Chaib-draa. Exact dynamic programming for decentralized POMDPs with lossless policy compression. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, pages 20–27, 2008.

[7] M. Chandrasekaran, P. Doshi, Y. Zeng, and Y. Chen. Can bounded and self-interested agents be teammates ? application to planning in ad hoc teams. *Autonomous Agents and Multi-Agent Systems*, 31(4) :821–860, Jul 2017.

[8] J. Cohen, J. S. Dibangoye, and A.-I. Mouaddib. Open Decentralized POMDPs. In *Proceedings of the 2017 International Conference on Tools for Artificial Intelligence*, 2017.

[9] I. Curiel. *Cooperative Game Theory and Applications : Cooperative Games Arising from Combinatorial Optimization Problems*. Theory and Decision Library C. Springer US, 2013.

[10] J. S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet. Optimally solving Dec-POMDPs as continuous-

state MDPs. *Journal of Artificial Intelligence Research*, 55 :443–497, 2016.

[11] A. Kumar and S. Zilberstein. Point-based backup for decentralized POMDPs : complexity and new algorithms. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pages 1315–1322, 2010.

[12] R. B. Myerson. Population uncertainty and Poisson games. *International Journal of Game Theory*, 27(3) :375–392, 1998.

[13] R. Nair, M. Tambe, M. Yokoo, D. V. Pynadath, and S. Marsella. Taming decentralized POMDPs : Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 705–711, 2003.

[14] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs : A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 133–139, 2005.

[15] F. A. Oliehoek and C. Amato. Dec-POMDPs as Non-Observable MDPs. *IAS technical reports*, 2014.

[16] F. A. Oliehoek, M. T. Spaan, and N. A. Vlassis. Optimal and approximate Q-value functions for Decentralized POMDPs. In *Journal of Artificial Intelligence Research*, volume 32, pages 289–353, 2008.

[17] F. A. Oliehoek, M. T. J. Spaan, C. Amato, and S. Whiteson. Incremental clustering and expansion for faster optimal planning in Dec-POMDPs. *Journal of Artificial Intelligence Research*, 46 :449–509, 2013.

[18] C. Papadimitriou and J. Tsitsiklis. The complexity of Markov decision processses. *Mathematics of Operations Research*, 12(3) :441–450, 1987.

[19] D. Ray and R. Vohra. Coalition formation. In *Handbook of game theory with economic applications*, volume 4, pages 239–326. Elsevier, 2015.

[20] S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2) :190–250, 2008.

[21] L. S. Shapley. Cores of convex games. *International Journal of Game Theory*, 1(1) :11–26, Dec 1971.

[22] M. T. Spaan, F. A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Sixth Annual Workshop on Multiagent Sequential Decision Making in Uncertain Domains (MSDM-2011)*, page 63. Citeseer, 2011.

[23] P. Stone, G. A. Kaminka, S. Kraus, J. R. Rosenschein, and N. Agmon. Teaching and leading an ad hoc teammate : Collaboration without pre-coordination. *Artificial Intelligence*, 203 :35–65, October 2013.

[24] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams : Collaboration without pre-coordination. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, July 2010.

[25] P. Varakantham, Y. Adulyasak, and P. Jaillet. Decentralized stochastic planning with anonymity in interactions. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Sur le Gradient de la Politique pour les Systèmes Multi-Agents Coopératifs

G. Bono[1] J. Dibangoye[1] L. Matignon[1,2] F. Pereyron[3] O. Simonin[1]

[1] Univ Lyon, INSA Lyon, INRIA, CITI, F-69621 Villeurbanne, France
[2] Univ Lyon, Université Lyon 1, LIRIS, CNRS, UMR5205, Villeurbanne, F-69622, France
[3] Volvo Group, Advanced Technology and Research

guillaume.bono@inria.fr

Résumé

L'apprentissage par renforcement (RL) pour les processus décisionnels de Markov partiellement observables décentralisés (Dec-POMDPs) accuse un certain retard par rapport aux progrès spectaculaires du RL mono-agent. Ceci s'explique en partie par un certain nombre d'hypothèses valables dans le cadre mono-agent, mais invalides dans les systèmes multi-agents. Pour combler ce retard, nous explorons les fondements mathématiques des méthodes par ascension du gradient de la politique dans le paradigme de l'entraînement centralisé pour un contrôle décentralisé (CTDC). Dans ce paradigme, l'apprentissage peut avoir lieu de façon centralisée tout en gardant la contrainte d'une exécution décentralisée. En partant de cette intuition, nous établissons dans ce document une extension multi-agents du théorème du gradient de la politique et du théorème de compatibilité des fonctions d'approximation de la valeur. Nous en tirons des méthodes « acteur critique » (AC) qui parviennent (i) à estimer le gradient de la politique à partir d'expériences collectives mais aussi (ii) à préserver le contrôle décentralisé du système à l'exécution. Nos expérimentations montrent que nos méthodes ne souffrent pas de la comparaison avec les techniques standard en RL sur un ensemble de bancs de test de la littérature.

Mots Clef

Contrôle décentralisé et stochastique – Processus Décisionnel de Markov Partiellement Observable – Systèmes Multi-Agents – Méthodes Acteur Critique

Abstract

Reinforcement Learning (RL) for decentralized partially observable Markov decision processes (Dec-POMDPs) is lagging behind the spectacular breakthroughs of single-agent RL. That is because assumptions that hold in single-agent settings are often obsolete in decentralized multi-agent systems. To tackle this issue, we investigate the foundations of policy gradient methods within the centralized training for decentralized control (CTDC) paradigm. In this paradigm, learning can be accomplished in a centralized manner while each agent can still execute its policy independently at deployment. Using this insight, we establish a new policy gradient theorem and compatible function approximations for decentralized multi-agent systems. Resulting actor critic methods preserve the decentralized control at the execution phase, but can also estimate the policy gradient from collective experiences guided by a centralized critic at the training phase. Experiments demonstrate our policy gradient methods compare favorably against standard RL techniques in benchmarks from the literature.

Keywords

Decentralized and Stochastic Control – Partially Observable Markov Decision Processes – Multi-Agent Systems – Actor Critic Methods.

1 Introduction

Ces dernières années, la capacité d'agents artificiels à apprendre des comportements par eux-mêmes en interagissant avec leur environnement s'est développée de façon spectaculaire [18, 19], promettant de grandes avancées dans la société et l'industrie. Une partie de ces progrès peut être imputée à l'apprentissage par renforcement (RL) mono-agent, particulièrement le RL profond. Il s'agit d'une branche de l'apprentissage automatique où le monde dans lequel évolue l'agent est décrit comme un processus décisionnel de Markov (MDP) [24]. Si d'autres agents interviennent, ils sont intégrés à cette description du monde, et les hypothèses considérées pendant l'apprentissage et l'exécution sont identiques. Dans ce modèle, les méthodes par ascension du gradient de la politique et les algorithmes *acteur critique* (AC) et *acteur critique naturel* (NAC) ont eu un certain succès, avec de bonnes garanties de convergence [1, 25, 14, 6]. Ces méthodes cherchent directement dans un espace de politiques stochastiques paramétrées en ajustant la valeur des paramètres dans la direction du gradient de la politique. Malheureusement, les adaptations de ces méthodes au cadre multi-agents se sont focalisées soit sur des agents indépendants [28, 21], soit sur des systèmes

où la connaissance du monde est commune [31]. Ces derniers peuvent en fait être ramenés à des systèmes mono-agent.

Au lieu de cela, nous considérons dans cet article un système multi-agents coopératif où l'apprentissage est centralisé, mais où l'exécution reste décentralisée. Grâce à ce paradigme, nous pouvons relâcher la contrainte de contrôle décentralisé pendant la phase d'apprentissage. Nous préservons néanmoins l'indépendance des politiques apprises pour leur exécution. Dans de nombreux cas d'application des systèmes multi-agents, les conditions lors de l'entraînement ne sont pas aussi strictes que les conditions à l'exécution : pendant les répétitions d'une pièce de théâtre, les acteurs peuvent lire le script, prendre des pauses ou échanger avec le metteur en scène ; pour gagner un match de football, les joueurs appliquent des tactiques développées avec leur entraîneur bien avant la rencontre, etc. Il est donc naturel de se demander si les méthodes d'ascension du gradient de la politique peuvent tirer profit de ce paradigme.

Le paradigme CTDC a été utilisé avec succès à des méthodes de planification pour les MDP partiellement observables décentralisés (Dec-POMDP), un modèle de choix pour la prise de décision séquentielle par un ensemble d'agents coopérant à la poursuite d'un objectif commun [4, 13, 26, 20, 10, 27, 8, 9]. Dans la littérature de la théorie des jeux, les Dec-POMDP sont des jeux stochastiques partiellement observables à gains identiques. Les Dec-POMDP généralisent d'autres modèles multi-agents collaboratifs, comme les MDP multi-agents [5], ou les jeux stochastiques à gains identiques [23] par exemple. L'hypothèse principale qui rend les Dec-POMDP fondamentalement différents et plus complexes que les MDP pourrait n'être pertinente que pendant la phase d'exécution : les agents n'ont pas accès au véritable état du monde, et ne peuvent pas transmettre aux autres agents leurs observations locales bruitées. Rien n'empêche les agents de partager leurs informations locales pendant l'entraînement à la condition qu'à l'exécution, ils ne basent leurs décisions que sur leurs expériences individuelles. De façon assez surprenante, cette intuition a été relativement peu exploitée, et le traitement formel du paradigme CTDC a fait l'objet de peu d'attention au sein de la communauté RL [16]. Quand cet entraînement centralisé a lieu en laboratoire ou se base sur un simulateur, il est possible d'exploiter l'information disponible plus riche que celle collectée lors de l'exécution, par exemple les états cachés, les informations locales d'autres agents, etc. Même si les travaux récents en RL multi-agents – et RL profond – partent de ce paradigme pour développer des méthodes spécifiques à certains domaines [12, 17, 11], les fondements théoriques du RL multi-agents décentralisé en sont encore à leurs balbutiements.

Cet article explore les fondements théoriques des méthodes du gradient de la politique dans le paradigme CTDC. Dans ce paradigme, les algorithmes acteur critique peuvent être adaptées pour entraîner plusieurs acteurs (ou politiques) indépendants guidés par un critique centralisé (c.à.d. une approximation de la fonction de valeur état-action) [11]. Les méthodes de cette famille diffèrent seulement par la façon dont est représenté et mis à jour ce critique centralisé. Le résultat principal de ce papier est une généralisation du théorème du gradient de la politique et du théorème de compatibilité des approximations de la valeur pour les Dec-POMDP en s'inspirant de leurs équivalents en MDP. Nous montrons en particulier que le critique centralisé est compatible avec le gradient de la politique s'il se décompose en une somme de critiques individuels, eux-mêmes combinaisons linéaires des caractéristiques de la politique individuelle correspondante. De plus, nous dérivons des règles de mise à jour qui ajustent les paramètres des critiques individuels dans la direction du gradient du critique centralisé. Nos expérimentations tendent à montrer que notre approche est compétitive par rapport aux techniques issues des paradigmes classiques du RL sur un jeu de bancs de test de la littérature.

Le reste de l'article est structuré de la façon suivante. La Section 2 revient sur la définition formelle des MDP partiellement observables (POMDP) et Dec-POMDP, et rappelle un ensemble de propriétés pertinentes. Dans la Section 3, nous passons en revue les méthodes du gradient de la politique pour les POMDP (mono-agent), puis nous poursuivons par une analyse des travaux les utilisant dans le cadre multi-agents coopératif en Section 4. La Section 5 présente nos résultats sur les fondements théoriques des méthodes du gradient de la politique pour les Dec-POMDP, et en dérive des algorithmes acteur-critique. Nous concluons par la présentation et l'analyse de nos résultats empiriques dans la Section 6

2 Contexte

2.1 MDP partiellement observables

Considérons un agent jouant le rôle de coordinateur central chargé de contrôler le comportement d'un processus décisionnel de Markov partiellement observable (POMDP) au cours de son évolution dans le temps. Ce cadre sert souvent à formaliser les systèmes multi-agents coopératifs dans lesquels tous les agents se transmettent explicitement et instantanément leurs observations bruitées.

Définition 1. *Soit* $M_1 \doteq (\mathcal{X}, \mathcal{U}, \mathcal{Z}, p, R, T, s_0, \gamma)$ *un POMDP. On note* X_t, U_t, Z_t *et* R_t *les variables aléatoires qui prennent respectivement leurs valeurs dans* $\mathcal{X}$, $\mathcal{U}$, $\mathcal{Z}$ *et* $\mathbb{R}$ *et qui décrivent : l'état de l'environnement, les commandes appliquées par l'agent, les observations et le signal de récompense qu'il a reçu à l'instant* $t = 0, 1, \ldots, T$. *La dynamique de l'environnement est modélisée par les probabilités de transition et d'observation* $p(x', z|x, u) \doteq \mathbb{P}(X_{t+1} = x', Z_{t+1} = z|X_t = x, U_t = u)$. $R(x, u) \doteq \mathbb{E}[R_{t+1}|X_t = x, U_t = u]$ *dénote l'espérance de la récompense immédiate,* $s_0(x) = \mathbb{P}(X_0 = x)$ *la distribution initiale sur les états, et* $\gamma \in [0, 1]$ *le facteur d'amortissement des récompenses futures.*

On définit récursivement l'historique $o_t \doteq (o_{t-1}, u_{t-1}, z_t)$

avec $o_0 \doteq \varnothing$, contenant la séquence de commandes et d'observations que l'agent a effectuées et perçues jusqu'au temps t. On note $\mathcal{O}_t$ l'ensemble des historiques potentiellement expérimentés par l'agent au temps t.

Définition 2. *L'agent sélectionne à chaque instant une commande u_t selon une politique paramétrée $\pi \doteq (a_0, a_1, \ldots, a_T)$, où $a_t(u_t|o_t) \doteq \mathbb{P}(u_t|o_t;\theta_t)$ dénote la règle de décision appliquée au temps t, de paramètres $\theta_t \in \mathbb{R}^{\ell_t}$ avec $\ell_t \ll |\mathcal{O}_t|$, $\forall t \in \{0 \ldots T\}$.*

En pratique, la politique peut être construite comme par exemple un réseau de neurone profond, un contrôleur à états finis ou encore une simple structure linéaire normalisée par *softmax*. De telles représentations de la politique reposent sur différentes descriptions parfois incomplètes des historiques, appelées états internes de l'agent. Notons que si le modèle est connu ou estimé, il peut servir à calculer une forme d'états internes appelés croyances qui constituent une statistique exhaustive de l'historique [2]. Si on note $b^o \doteq \mathbb{P}(X_t|O_t = o)$ la croyance courante induit par l'historique o, avec une croyance initial $b^{\varnothing} \doteq s_0$, alors, après avoir effectué la commande $u \in \mathcal{U}$ et perçu l'observation $z \in \mathcal{Z}$, la nouvelle croyance se calcule ainsi : $\forall x' \in \mathcal{X}$,

$$\begin{aligned} b^{o,u,z}(x') &\doteq \mathbb{P}(X_{t+1} = x'|O_{t+1} = (o,u,z)) \\ &\propto \sum_{x\in\mathcal{X}} p(x', z|x, u) b^o(x). \end{aligned}$$

Par conséquent, utiliser les croyances au lieu des historiques dans la description des politiques préserve la capacité à agir de façon optimale, tout en réduisant significativement l'empreinte mémoire de la représentation de l'état interne. Cela permet aussi de se focaliser sur des politiques stationnaires, particulièrement intéressantes dans le cadre à horizon infini ($T = \infty$). Une politique π est dite stationnaire si $a_0 = a_1 = \ldots = a$, ou dit autrement $\theta_0 = \theta_1 = \ldots = \theta$; sinon, elle est non stationnaire.
En évoluant dans l'environnement guidé par la politique π, l'agent génère une trajectoire composée de récompenses, d'observations, de commandes et d'états $\omega_{t:T} \doteq (r_{t:T}, x_{t:T}, z_{t:T}, u_{t:T})$. Chaque trajectoire rapporte une récompense cumulée $R(\omega_{t:T}) \doteq \gamma^0 r_t + \cdots + \gamma^{T-t} r_T$. Les meilleures politiques sont celles qui produisent à partir de s_0 les meilleures récompense cumulée en espérance :

$$\begin{aligned} J(s_0;\theta_{0:T}) &\doteq \mathbb{E}_{\Omega_{0:T}\sim\mathbb{P}(\cdot|\pi,M_1)}[R(\Omega_{0:T})] \\ &= \sum_{\omega_{0:T}} \mathbb{P}(\omega_{0:T}|\pi, M_1) R(\omega_{0:T}) \end{aligned} \quad (1)$$

où on note $\mathbb{P}(\omega_{0:T}|\pi, M_1)$ la probabilité de générer la trajectoire $\omega_{0:T}$ en suivant π. Trouver le meilleur moyen pour l'agent d'interagir avec M_1 consiste à trouver le vecteur de paramètres $\theta^*_{0:T}$ solution de : $\theta^*_{0:T} \in \arg\max_{\theta_{0:T}} J(s_0;\theta_{0:T})$.
Il s'avère judicieux de subdiviser la performance d'une politique pour exploiter sa structure sous-jacente : en effet, les performances d'une politique π à partir du temps t ne dépendent des commandes antérieures à t qu'à travers les états et historiques courants. Cela nous conduit à définir les fonctions de valeur, Q-valeur et avantage de π.
La fonction de Q-valeur de π est défini par :

$$Q^\pi_t : (x,o,u) \mapsto \mathbb{E}[R(\Omega_{t:T})] \quad (2)$$

où $\Omega_{t:T} \sim \mathbb{P}(\cdot|X_t = x, O_t = o, U_t = u; \pi, M_1)$.
$Q^\pi_t(x,o,u)$ est l'espérance sur la récompense cumulée en exécutant au temps t la commande u à partir d'un état x et d'un historique o, puis en choisissant par π les futures commandes à partir de $t+1$. La fonction de valeur de π est donnée par :

$$V^\pi_t : (x,o) \mapsto \mathbb{E}_{U\sim a_t(\cdot|o)}[Q^\pi_t(x,o,U)] \quad (3)$$

où $V^\pi_t(x,o)$ est l'espérance sur la récompense cumulée en suivant la politique π à partir du temps t depuis un état x et un historique o. Enfin, la fonction avantage de π est simplement la différence des deux :

$$A^\pi_t : (x,o,u) \mapsto Q^\pi_t(x,o,u) - V^\pi_t(x,o) \quad (4)$$

où $A^\pi_t(x,o,u)$ est donc l'avantage relatif à exécuter u plutôt que de suivre π au temps t depuis un état x et un historique o, puis en revenant à la politique π par la suite. La propriété qui rend ces fonctions intéressantes est qu'elles vérifient la relation de récurrence suivante.

Lemme 1 (Équations de Bellman [3])**.** $\forall t = 0, 1, \ldots, T$, $\forall x \in \mathcal{X}, o \in \mathcal{O}_t, u \in \mathcal{U}$,

$$Q^\pi_t(x,o,u) = R(x,u) + \gamma \mathbb{E}[Q^\pi_{t+1}(X', O', U')] \quad (5)$$

où $(X', O', U') \sim \mathbb{P}(\cdot|X_t = x, O_t = o, U_t = u; a_{t+1}, M_1)$

(5) établit une relation temporelle entre les différents $V^\pi_{0:T}$, $Q^\pi_{0:T}$ et $A^\pi_{0:T}$, mais aussi $J(s_0;\theta_{0:T})$:

$$J(s_0;\theta_{0:T}) = \mathbb{E}_{X_0\sim\mathbb{P}(\cdot|s_0)}[V^\pi_0(X_0, o_0 = \varnothing)]. \quad (6)$$

Jusqu'à présent, nous nous sommes concentrés sur les systèmes contrôlés par un unique agent. La suite généralise aux systèmes décentralisés dans lesquels plusieurs agents doivent coopérer pour contrôler le même système.

2.2 POMDP Décentralisés

Considérons à présent un contexte où n agents coopèrent pour influencer l'évolution d'un système décrit comme un POMDP, mais où aucun d'eux ne peut percevoir l'état du monde, ni communiquer ses observations bruitées aux autres.

Définition 3. *Un Dec-POMDP $M_n \doteq (\mathcal{I}_n, \mathcal{X}, \mathcal{U}, \mathcal{Z}, p, R, T, \gamma, s_0)$ est défini tel que : $i \in \mathcal{I}_n$ dénote l'indice du $i^{ème}$ agent impliqué dans le processus ; $\mathcal{X}, \mathcal{U}, \mathcal{Z}, p, R, T, \gamma$ et s_0 sont définis tels que dans M_1, à la particularité que $\mathcal{U}$ et $\mathcal{Z}$ se décomposent en ensembles de commandes et d'observations individuelles $\mathcal{U} = \mathcal{U}^1 \times \cdots \times \mathcal{U}^n$ et $\mathcal{Z} = \mathcal{Z}^1 \times \cdots \times \mathcal{Z}^n$, avec $\mathcal{U}^i$ et $\mathcal{Z}^i$ l'ensemble des commandes et des observations propres au $i^{ème}$ agent.*

On appelle historique individuel de l'agent $i \in \mathcal{I}_n$ la séquence de commandes et d'observations individuelles exécutées et perçues jusqu'au temps $t = 0, 1, \ldots, T$, notée $o_t^i = (o_{t-1}^i, u_{t-1}^i, z_t^i)$ avec $o_0^i = \varnothing$. On note $\mathcal{O}_t^i$, l'ensemble des historiques individuels possible de l'agent i au temps t.

Définition 4. *L'agent $i \in \mathcal{I}_n$ sélectionne la commande u_t^i au temps t selon une politique paramétrée $\pi^i \doteq (a_0^i, a_1^i, \ldots, a_T^i)$ où $a_t^i(u_t^i|o_t^i) \doteq \mathbb{P}(u_t^i|o_t^i; \theta_t^i)$ est la règle de décision appliquée au temps t, de paramètres $\theta_t^i \in \mathbb{R}^{\ell_t^i}$, avec $\ell_t^i \ll |\mathcal{O}_t^i|$.*

De même que dans M_1, le nombre d'historiques individuels possibles grandit de façon exponentiel à chaque pas de temps. À ce jour, la seule statistique exhaustive connue pour les historiques individuels repose sur les états d'occupation définis par : $s_t(x, o) \doteq \mathbb{P}(x, o|\theta_{0:t-1}^{1:n}), \forall x \in \mathcal{X}, \forall o \in \mathcal{O}_t$. L'état d'occupation individuel induit par l'historique $o^i \in \mathcal{O}_t^i$ est une distribution de probabilité conditionnelle : $s_t^i(x, o^{-i}) \doteq \mathbb{P}(x, o^{-i}|o^i, s_t)$, où o^{-i} dénote l'historique joint des $n-1$ agents autres que i. Il est très difficile d'apprendre à projeter les historiques individuels vers des états internes proches des états d'occupation individuels, ce qui limite la capacité des algorithmes d'apprentissage à trouver des politiques optimales dans un temps raisonnable pour M_n. On peut alors se focaliser sur des politiques stationnaires pour lesquelles l'espace des historiques est projeté sur un ensemble fini d'états internes $\varsigma \doteq (\varsigma^1, \ldots, \varsigma^n)$, qui sont des représentations – souvent à pertes – des états d'occupations individuels. Les ς^i peuvent par exemple être les nœuds d'un contrôleur à états finis, ou encore les états internes d'un réseau de neurone récurrent (RNN). On note $\psi(\varsigma'|\varsigma, u, z)$ la probabilité de transition d'un état interne joint ς à l'état interne suivant ς' en fonction de la dernière commande effectuée u et de l'observation reçue z. Cette loi de transition est décomposable en lois individuelles $\psi^i(\varsigma'^{,i}|\varsigma^i, u^i, z^i)$. Dans la suite de cet article, nous considérerons la loi ψ fixée à priori, même si en général, elle fait partie de la politique et possède des paramètres à optimiser. Résoudre M_n revient à trouver une politique jointe $\pi \doteq (\pi^1, \ldots, \pi^n)$ – un tuple de n politiques individuelles – qui donne la meilleure récompense cumulée à partir d'une distribution d'états initiaux s_0. Autrement dit, $\theta_{0:T}^{*,1:n} \in \arg\max_{\theta_{0:T}^{1:n}} J(s_0; \theta_{0:T}^{1:n})$, avec

$$\begin{aligned} J(s_0; \theta_{0:T}^{1:n}) &\doteq \mathbb{E}_{\Omega_{0:T} \sim \mathbb{P}(\cdot|\pi, M_n)}[R(\Omega_{0:T})] \\ &= \sum_{\omega_{0:T}} \mathbb{P}(\omega_{0:T}|\pi, M_n) R(\omega_{0:T}) \end{aligned} \tag{7}$$

où $\mathbb{P}(\omega_{0:T}|\pi, M_n)$ est la probabilité de générer la trajectoire jointe $\omega_{0:T}$ en suivant la politique π. Étant donné une politique jointe π, M_n hérite des mêmes définitions que M_1, en particulier pour les fonctions $V_{0:T}^\pi$, $Q_{0:T}^\pi$ et $A_{0:T}^\pi$.

3 Gradient de la politique

Dans cette section, nous passons en revue la littérature des méthodes du gradient de la politique pour le contrôle centralisé de systèmes mono-agent. Dans ce contexte, l'approche du gradient de la politique se base sur des algorithmes centralisés qui cherchent à optimiser la récompense cumulée directement dans l'espace des paramètres de la politique $\theta_{0:T}$. Bien que nous n'abordions ici que des politiques non stationnaires, les méthodes citées peuvent facilement être étendues au cas de politiques stationnaires pour lesquelles $a_t = a$ pour tout t. En considérant que la politique π est différentiable par rapport à ses paramètres $\theta_{0:T}$, l'algorithme met à jour $\theta_{0:T}$ dans la direction du gradient :

$$\Delta\theta_{0:T} = \alpha \frac{\partial J(s_0; \theta_{0:T})}{\partial \theta_{0:T}} \tag{8}$$

avec α un taux d'apprentissage. En itérant sur cette règle de mise à jour, pour peu que l'estimation du gradient soit correcte, $\theta_{0:T}$ converge vers un optima local. Malheureusement, estimer correctement le gradient peut se révéler impossible. Pour dépasser cette difficulté, le gradient peut être remplacé par une estimation non biaisée, ce qui revient à restreindre (8) à un gradient stochastique :

$$\Delta\theta_{0:T} = \alpha R(\omega_{0:T}) \frac{\partial \log \mathbb{P}(\omega_{0:T}|\pi, M_1)}{\partial \theta_{0:T}} \tag{9}$$

On calcule $\frac{\partial}{\partial \theta_{0:T}} \log \mathbb{P}(\omega_{0:T}|\pi, M_1)$ sans connaissance à priori de la distribution sur les trajectoires $\mathbb{P}(\omega_{0:T}|\pi, M_1)$. En effet, les propriétés d'indépendance conditionnelle du processus donnent :

$$\mathbb{P}(\omega_{0:T}|\pi, M_1) \doteq s_0(x_0) \prod_{t=0}^{T} p(x_{t+1}, z_{t+1}|x_t, u_t) a_t(u_t|o_t) \tag{10}$$

ce qui implique :

$$\frac{\partial \log \mathbb{P}(\omega_{0:T}|\pi, M_1)}{\partial \theta_{0:T}} = \sum_{t=0}^{T} \frac{\partial \log a_t(u_t|o_t)}{\partial \theta_t} \tag{11}$$

3.1 Méthode du rapport de vraisemblance

Les méthodes du rapport de vraisemblances, *e.g.,* `Reinforce` [29], exploitent la séparabilité du vecteur de paramètres $\theta_{0:T}$, qui mènent à la règle de mise à jour suivante : $\forall t = 0, 1, \ldots, T$

$$\Delta\theta_t = \alpha \mathbb{E}_{\mathcal{D}} \left[R(\omega_{0:T}) \frac{\partial \log a_t(u_t|o_t)}{\partial \theta_t} \right] \tag{12}$$

où $\mathbb{E}_{\mathcal{D}}[\cdot]$ dénote une approximation de l'espérance par une moyenne empirique prise sur un ensemble de trajectoires $\mathcal{D} = \{\omega_{0:T,j}\}_{1 \leqslant j \leqslant m}$ échantillonnées à partir de π et M_1. Le principal problème rencontré en utilisant cette règle de mise à jour centralisé est la variance importante de $R(\omega_{0:T})$, qui peut fortement ralentir la convergence. Pour compenser partiellement cette variance, deux constatations rapides permettent d'améliorer la règle de mise à jour. Premièrement, les futures actions ne dépendent pas des récompenses passées, *c.à.d.* $\mathbb{E}_{\mathcal{D}}[R(\omega_{0:t-1}) \frac{\partial}{\partial \theta_t} \log a_t(u_t|o_t)] =$

0, On peut donc utiliser $R(\omega_{t:T})$ au lieu de $R(\omega_{0:T})$ dans 12, ce qui réduit significativement la variance de notre estimation du gradient. Deuxièmement, l'estimation du gradient de la politique n'est pas biaisée par l'introduction d'une fonction de référence $\tilde{\beta}_t$ (en anglais : *baseline*) tant que celle-ci ne dépend pas de $\theta_{0:T}$. On peut ainsi remplacer $R(\omega_{t:T})$ par une récompense relative à cette référence $R(\omega_{t:T}) - \tilde{\beta}_t(x_t, o_t)$.

3.2 Acteur Critique

Pour améliorer la variance de l'estimation du gradient dans (12), le théorème du gradient de la politique [25] propose de remplacer $R(\omega_{t:T})$ par $Q_t^w(x_t, o_t, u_t)$, une approximation de la Q-valeur associée à la commande u_t dans l'état x_t et après un historique o_t puis en complétant la trajectoire en suivant la politique π : $Q_t^w(x_t, o_t, u_t) \approx Q_t^\pi(x_t, o_t, u_t)$, où $w_t \in \mathbb{R}^{l_t}$ est un vecteur de paramètres, avec $l_t \ll |\mathcal{X}||\mathcal{O}_t||\mathcal{U}|$. Ceci conduit au schéma algorithmique *acteur critique*, dans lequel un algorithme centralisé met à jour à la fois les paramètres $\theta_{0:T}$ de la politique et les paramètres $w_{0:T}$ de l'approximation de la Q-valeur :

$$\Delta w_t = \alpha \mathbb{E}_{\mathcal{D}} \left[\delta_t \frac{\partial \log a_t(u_t|o_t)}{\partial \theta_t} \right] \tag{13a}$$

$$\Delta \theta_t = \alpha \mathbb{E}\mathcal{D} \left[Q_t^w(x_t, o_t, u_t) \frac{\partial \log a_t(u_t|o_t)}{\partial \theta_t} \right] \tag{13b}$$

où on note $\delta_t \doteq \widehat{Q}_t^\pi(x_t, o_t, u_t) - Q_t^w(x_t, o_t, u_t)$, avec $\widehat{Q}_t^\pi(x_t, o_t, u_t)$ une estimation non-biaisée de la véritable Q-valeur $Q_t^\pi(x_t, o_t, u_t)$, utilisant par exemple une estimation Monte-Carlo (MC) de $R(\omega_{0:T})$ ou bien une différence temporelle (erreur TD) $r_t + \gamma Q_{t+1}^w(x_{t+1}, o_{t+1}, u_{t+1})$. Le choix des paramètres $w_{0:T}$ est crucial pour que l'estimation du gradient reste non-biaisé [25]. Il n'y a pas de biais si les approximations de Q-valeur $Q_{0:T}^w$ sont *compatibles* avec la politique π. Sans rentrer dans les détails formels, une approximation compatible $Q_{0:T}^w$ de la véritable fonction $Q_{0:T}^\pi$ peut être une combinaison linéaire des « caractéristiques » de la politique π, et ses paramètres $w_{0:T}$ constituent la solution du problème de régression linéaire qui estime $Q_{0:T}^\pi$ à partir de ces caractéristiques. En pratique, cette seconde condition n'est pas utilisée directement, et les paramètres $w_{0:T}$ sont mis à jour en utilisant par exemple des méthodes MC ou TD.

3.3 Acteur Critique Naturel

Suivre la direction du gradient n'est pas toujours le moyen le plus rapide pour converger. Le gradient naturel propose une mise à jour des paramètres $\theta_{0:T}$ selon la direction de plus rapide ascension par rapport à la métrique d'information de Fisher :

$$\mathbf{\Phi} \doteq \mathbb{E}_{\mathcal{D}} \left[\frac{\partial \log a_t(u_t|o_t)}{\partial \theta_t} \left(\frac{\partial \log a_t(u_t|o_t)}{\partial \theta_t} \right)^\top \right] \tag{14}$$

Cette métrique est invariante à une reparamétrisation de la politique. Combiner le théorème du gradient de la politique avec une approximation compatible de la Q-valeur, puis effectuer une mise à jour des paramètres dans la direction donnée par (14), donne naissance au schéma algorithmique *acteur-critique naturel*, qui remplace (13b) dans (13) par : $\Delta\theta_t = \alpha \mathbb{E}_{\mathcal{D}}[w_t]$.

4 Gradient de la politique pour les systèmes multi-agents

Dans cette section, nous passons en revue les extensions des méthodes du gradient de la politique mono-agent à des modèles multi-agents coopératifs. On distingue trois paradigmes, illustrés dans la Figure 1 : l'entraînement centralisé pour un contrôle centralisé (CTCC), l'entraînement distribué pour un contrôle décentralisé (DTDC), et l'entraînement centralisé pour un contrôle décentralisé (CTDC).

4.1 CTCC

Dans certaines applications des systèmes multi-agents coopératifs, on peut considérer des agents pouvant communiquer entre eux de façon parfaite, instantanée et illimitée. De telles applications peuvent être modélisées par des POMDP, ce qui rend possible d'utiliser les méthodes du gradient de la politique mono-agent (Section 3). Dans ce paradigme CTCC, illustré par la Figure 1 *(gauche)*, un seul acteur « coordinateur » guidé par un critique unique est utilisé. Le principal défaut de ce paradigme fait aussi sa force : ce besoin de communication parfaite, instantanée et illimitée entre les agents d'un bout à l'autre de l'exécution, à la fois pendant l'entraînement et le déploiement.

4.2 DTDC

Assez étonnamment, il s'avère que la première adaptation des méthodes du gradient de la politique au cadre multi-agent visait à apprendre de façon distribuée les politiques à exécuter de façon décentralisée, comme par exemple l'algorithme `Reinforce` distribué [21]. Dans ce paradigme DTDC, les agents apprennent simultanément mais indépendamment leurs politiques individuelles en utilisant l'algorithme Reinforce avec chacun un critique et un acteur individuels, comme l'illustre la Figure 1 *(droite)*. L'indépendance des vecteurs de paramètres $\theta_{0:T}^1, \theta_{0:T}^2, \ldots, \theta_{0:T}^n$ donne lieu à la règle de mise à jour suivante : $\forall t = 0, 1, \ldots, T, \forall i \in \mathcal{I}_n$,

$$\Delta\theta_t^i = \alpha \mathbb{E}_{\mathcal{D}} \left[R(\omega_{0:T}) \frac{\partial \log a_t^i(u_t^i|o_t^i)}{\partial \theta_t^i} \right] \tag{15}$$

Il est à noter que la somme des gradients des politiques individuelles est un estimateur non-biaisé du gradient de la politique jointe. Néanmoins, la question reste ouverte de savoir comment tirer profit des méthodes acteur-critique (voir Section 3) pour réduire la variance de cet estimateur joint. De plus, l'algorithme `Reinforce` distribué est restreint à un apprentissage où l'échantillonnage des trajectoires est dicté par la politique apprise π (*on-policy*), quand des politiques d'exploration mieux choisies $\bar{\pi}$ peuvent for-

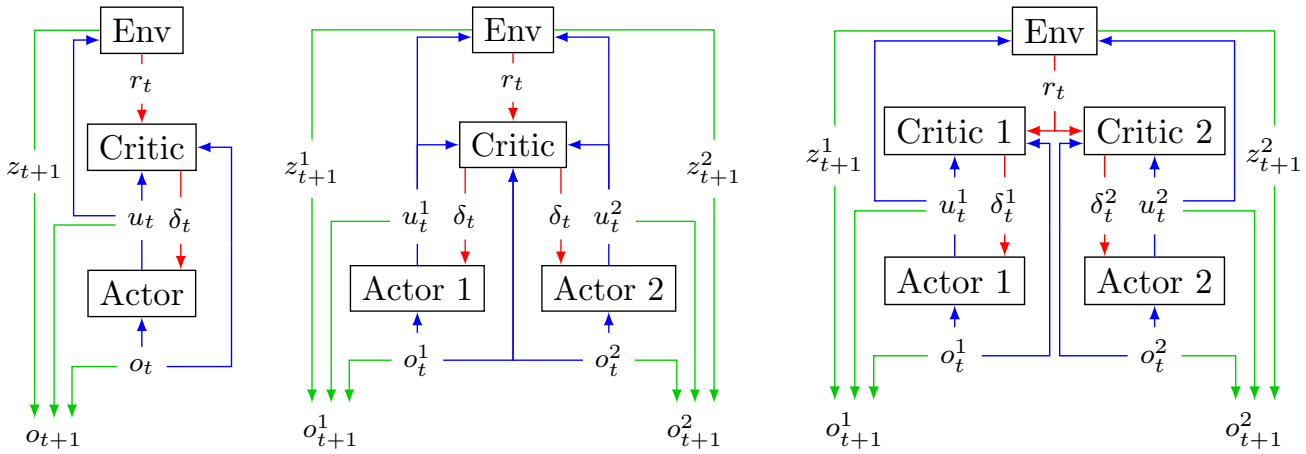


FIGURE 1 – Schéma algorithmique acteur-critique dans un modèle à deux agents pour les paradigmes : (*gauche*) CTCC, (*centre*) CTDC, et (*droite*) DTDC. Pour chaque figure, les flèches bleues représentent les commandes des agents sur l'environnement, les vertes montrent l'agrégation d'information pour le prochain pas de décision, et les rouges indiquent le signal de récompense propagé pour mettre à jour les différents paramètres.

tement améliorer la qualité de l'optimum local atteint à la convergence [6].

4.3 CTDC

Le paradigme CTDC a été employé avec succès pour résoudre des problèmes de planification de Dec-POMDP [4, 13, 26, 20, 10, 16, 27, 8, 9]. Dans un tel paradigme, un coordinateur central apprend pour l'ensemble des agents pendant la phase d'entraînement puis assigne les politiques individuelles apprises aux agents correspondants avant le début de la phase d'exécution. Comme le montre la Figure 1 *(centre)*, les algorithmes acteur critique dans ce paradigme CTDC mettent à jour un critique central mais optimisent un acteur par agent. Des travaux récents en apprentissage par renforcement (profond) [12, 17, 11] exploitent ce paradigme, en particulier l'approche par gradient de la politique multi-agents contrefactuel, et l'algorithme `COMA` qui en découle. Malheureusement, cette algorithme se focalise sur un critique central apportant une solution au problème de l'assignation des crédits. Pour un Dec-POMDP M_n (faiblement) séparable [7], il est possible d'apprendre une contribution relative de chaque agent à la valeur de la politique jointe, apportant ainsi une réponse au problème d'assignation du crédit. En revanche, en général, la dynamique et les récompenses d'un Dec-POMDP peuvent être fortement corrélées, ce qui rend difficile voire impossible d'accorder du crédit à tel ou tel agent. Contrairement aux méthodes de planification, les méthodes d'ascension du gradient de la politique dans ce paradigme CTDC manquent de fondements théoriques, qui permettraient de définir la forme du critique centralisé et les règles de mise à jour préservant certaines garanties formelles.

5 Gradient de la politique pour les Dec-POMDP

Dans cette section, nous apportons une réponse aux limitations des paradigmes CTCC et DTDC, et étendons les schémas algorithmiques acteur-critique et acteur-critique naturel de M_1 vers M_n.

5.1 Théorème du Gradient de la Politique

Notre principal résultat est une extension du théorème du Gradient de la politique [25] de M_1 vers M_n. Avant de poursuivre, nous commençons par exposer quelques résultats préliminaires qui nous permettrons d'établir notre résultat principal de la section. Toutes les preuves sont disponibles en annexes.

Lemme 2. *Pour toute fonction séparable* $f\colon (x^1,\dots,x^n) \mapsto f^1(x^1)\cdots f^n(x^n)$, *sa dérivée partielle par rapport à n'importe lequel de ses arguments* x^j, $j \in I_n$ *peut s'écrire :* $\frac{\partial}{\partial x^j} f(x) = f(x)\frac{\partial}{\partial x^j}\log f^j(x^j)$. *pour tout* $x = (x^1,\dots,x^n)$ *où* f *est différentiable et non nulle.*

Lemme 3. *Soit deux distributions* p *et* q *portant sur la même variable aléatoire* X. *Si le support de* p *est inclus dans celui de* q *alors :* $\mathbb{E}_{X\sim p}[f(X)] = \mathbb{E}_{X\sim q}[\frac{p(X)}{q(X)} f(X)]$.

Nous établissons à présent une expression des dérivées partielles des fonctions de valeurs $V^{\pi}_{0:T}$ par rapport aux vecteurs de paramètres $\theta^{1:n}_{0:T}$ dans le cas à horizon fini.

Lemme 4. *Soit un Dec-POMDP* M_n, *une politique jointe à évaluer* $\pi \doteq (a_0,\dots,a_T)$ *et une politique d'exploration* $\bar{\pi} \doteq (\bar{a}_0,\dots,\bar{a}_T)$. *Pour tout temps* $t = 0,1,\dots,T$, *agent*

$i \in \mathcal{I}_n$, état caché $x_t \in \mathcal{X}$, et historique joint $o_t \in \mathcal{O}_t$:

$$\frac{\partial V_t^\pi(x_t, o_t)}{\partial \theta_t^i} = \mathbb{E}_{U_t \sim \bar{a}_t(\cdot|o_t)}\Big[\frac{a_t(U_t|o_t)}{\bar{a}_t(U_t|o_t)} Q_t^\pi(x_t, o_t, U_t)\frac{\partial \log a_t^i(U_t^i|o_t^i)}{\partial \theta_t^i}\Big]. \quad (16)$$

Voici à présent le résultat principal de la section.

Théorème 1. *Soit un Dec-POMDP M_n, une politique à évaluer $\pi \doteq (a_0, \dots, a_T)$ et une politique d'exploration $\bar{\pi} \doteq (\bar{a}_0, \dots, \bar{a}_T)$.*

1. *Dans le cas à horizon fini $T < \infty$, pour tout temps $t = 0, 1, \dots, T$ et agent $i \in \mathcal{I}_n$:*

$$\frac{\partial J(s_0; \theta_{0:T}^{1:n})}{\partial \theta_t^i} = \gamma^t \mathbb{E}_{(X_t, O_t, U_t) \sim \mathbb{P}(\cdot|\bar{a}_t, M_n)}\Big[\frac{a_t(U_t|O_t)}{\bar{a}_t(U_t|O_t)} Q_t^\pi(X_t, O_t, U_t)\frac{\partial \log a_t^i(U_t^i|O_t^i)}{\partial \theta_t^i}\Big]$$

2. *Dans le cas à horizon infini $T = \infty$, pour tout agent $i \in \mathcal{I}_n$:*

$$\frac{\partial J(s_0; \theta^{1:n})}{\partial \theta^i} = \mathbb{E}_{(X, \Sigma, U) \sim \mathbb{P}(\cdot|\bar{s}, \bar{a})}\Big[\frac{a(U|\Sigma)}{\bar{a}(U|\Sigma)} Q^\pi(X, \Sigma, U)\frac{\partial \log a^i(U^i|\Sigma^i)}{\partial \theta^i}\Big]$$

où

$$\bar{s}(x, \varsigma) \doteq \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(X_t = x, \Sigma_t = \varsigma | M_n, \bar{a}, \psi, X_0, \varsigma_0)$$

Le théorème du gradient de la politique pour M_n (Théorème 1) est fondamentalement différent de celui pour M_1 [25]. Ce dernier part de l'hypothèse qu'un unique agent apprend à agir dans un (PO)MDP. À l'inverse, le Théorème 1 s'applique à un ensemble d'agents apprenant à contrôler l'évolution d'un POMDP de façon décentralisée. Les agents agissent indépendamment, mais leur estimation du gradient de la politique est guidée par une évaluation centralisée des fonctions $Q_{0:T}^\pi$. Pour utiliser cette propriété en pratique, il faut remplacer la véritable Q-valeur $Q_{0:T}^\pi$ par une approximation. Pour garantir que cette approximation est compatible – *c.à.d.* que le gradient calculé avec l'approximation est toujours dans la même direction que le véritable gradient – il faut lui donner une structure particulière que nous détaillons ci-dessous.

5.2 Approximations compatibles

Le résultat principal de cette section caractérise la forme des approximations compatibles $V_{0:T}^\sigma$ et $A_{0:T}^\nu$ respectivement pour les fonctions de valeur $V_{0:T}^\pi$ et les fonctions d'avantage $A_{0:T}^\pi$ pour un Dec-POMDP quelconque M_n. Ensemble, ces deux approximations permettent d'évaluer $Q_{0:T}^\pi(x_t, o_t, u_t) \doteq V_t^\pi(x_t, o_t) + A_t^\pi(x_t, o_t, u_t)$, pour tout temps $t = 0, 1, \dots, T$, historique joint $o_t \in \mathcal{O}_t$ et commande jointe $u_t \in \mathcal{U}$.

Théorème 2. *Soit un Dec-POMDP M_n, des fonctions d'approximation $V_{0:T}^\sigma$ et $A_{0:T}^\nu$, de paramètres respectifs $\sigma_{0:T}^{1:n}$ et $\nu_{0:T}^{1:n}$. Ces approximations sont compatibles avec la politique jointe $\pi \doteq (a_0, \dots, a_T)$ paramétrée par $\theta_{0:T}^{1:n}$ si l'une des deux conditions suivantes est vérifiée $\forall t = 0, 1, \dots, T$:*

1. *$\forall i \in \mathcal{I}_n, \forall x_t \in \mathcal{X}, \forall o_t \in \mathcal{O}_t$,*

$$\frac{\partial V_t^\sigma(x_t, o_t)}{\partial \sigma_t^i} = \mathbb{E}_{U_t^i \sim a_t^i(\cdot|o_t^i)}[\frac{\partial \log a_t^i(U_t^i|o_t^i)}{\partial \theta_t^i}] \quad (17)$$

   et σ *minimise l'erreur quadratique moyenne (MSE)* $\mathbb{E}[\epsilon_t(X_t, O_t, U_t)^2]$

2. *$\forall i \in \mathcal{I}_n, \forall x_t \in \mathcal{X}, \forall o_t \in \mathcal{O}_t, \forall u_t \in \mathcal{U}$,*

$$\frac{\partial A_t^\nu(x_t, o_t, u_t)}{\partial \nu_t^i} = \frac{\partial \log a_t^i(u_t^i|o_t^i)}{\partial \theta_t^i} \quad (18)$$

   et ν *minimise l'erreur quadratique moyenne (MSE)* $\mathbb{E}[\epsilon_t(X_t, O_t, U_t)^2]$

où $\epsilon_t(x_t, o_t, u_t) \doteq Q_t^\pi(x_t, o_t, u_t) - V_t^\sigma(x_t, o_t) - A_t^\nu(x_t, o_t, u_t)$.

Quand l'une de ces conditions est vérifiée, on a bien, pour toute politique d'exploration $\bar{\pi} \doteq (\bar{a}_0, \dots, \bar{a}_T)$:

$$\frac{\partial V_t^\pi(x_t, o_t)}{\partial \theta_t^i} = \mathbb{E}_{U_t \sim \bar{a}_t(\cdot|o_t)}\Big[\frac{a_t(U_t|o_t)}{\bar{a}_t(U_t|o_t)} \big(V_t^\sigma(x_t, o_t) + A_t^\nu(x_t, o_t, U_t)\big)\frac{\partial \log a_t^i(U_t^i|o_t^i)}{\partial \theta_t^i}\Big] \quad (19)$$

Le Théorème 2 est énoncé ci-dessus pour des politiques non-stationnaires et pour des problème à horizon fini $T < \infty$. Le résultat s'étend néanmoins très naturellement au cas à horizon infini et pour des politiques stationnaires où $a_t^i = a^i, \theta_t^i = \theta^i \ \forall t = 0, 1, \dots, \infty, \forall i \in \mathcal{I}_n$. Le théorème montre comment les conditions de compatibilité d'une approximation de la Q-valeur pour M_1 se généralisent dans le cas d'un Dec-POMDP M_n. Parmi les propriétés notables de ces approximations centralisées compatibles, il faut relever leur séparabilité :

$$V_t^\sigma : (x_t, o_t) \mapsto \sum_{i \in I_n} \mathbb{E}\left[\frac{\partial \log a_t^i(U_t^i|o_t^i)}{\partial \theta_t^i}\right]^\top \sigma_t^i \quad (20)$$

dans le premier cas, ou alors

$$A_t^\nu : (x_t, o_t, u_t) \mapsto \sum_{i \in I_n} \left(\frac{\partial \log a_t^i(u_t^i|o_t^i)}{\partial \theta_t^i}\right)^\top \nu_t^i \quad (21)$$

dans le second. Quel que soit le cas, à l'instar des résultats existants pour M_1, l'approximation A^ν ou V^σ dont la structure n'est pas contrainte constitue un degré de liberté très important pour tenter de réduire la variance de l'estimation du gradient. Dans notre paradigme CTDC, elle peut entre autre accéder aux état cachés x_t, ou encore combiner de façon plus intriquée l'information jointe disponible. Par

ailleurs, il est tout à fait envisageable d'exploiter la séparabilité des deux approximations compatibles :

$$Q_t^\pi(x_t, o_t, u_t) \approx \sum_{i \in I_n} \left(\frac{\partial \log a_t^i(u_t^i|o_t^i)}{\partial \theta_t^i} \right)^\top \nu_t^i + \sum_{i \in I_n} \mathbb{E}\left[\frac{\partial \log a_t^i(U_t^i|o_t^i)}{\partial \theta_t^i} \right]^\top \sigma_t^i + \tilde{\beta}_t(x_t, o_t, u_t) \quad (22)$$

avec $\tilde{\beta}_t$ une fonction de référence arbitraire, ne dépendant ni de ν, ni de σ, ni de θ et ne perturbant donc pas le gradient. Notons enfin que la séparabilité du critique centralisé ne nous permet pas de considérer les critiques individuels indépendamment l'un de l'autre, l'approximation du gradient est toujours jointe et guidée par cet unique critique centralisé.

5.3 Algorithmes Acteur Critique pour le Contrôle Décentralisé

Dans cette section, nous utilisons les résultats du Théorème 2 pour donner naissance à un schéma algorithmique acteur-critique pour M_n adapté au paradigme CTDC, que nous appelons *acteur critique pour le contrôle décentralisé* (ACDC). Cet algorithme ne nécessite pas de connaissance à priori du modèle, et l'échantillonnage des trajectoires peut se servir d'une politique d'exploration différente de la politique à évaluer. Il est centralisé [1] et itératif. Chaque itération consiste en une étape d'évaluation de la politique apprise, suivi d'une étape d'amélioration de celle-ci. L'étape d'évaluation crée une base d'échantillons de trajectoires (mini-batch) $\mathcal{D}$ à partir de $\mathbb{P}(\Omega_{0:T}|\pi, M_n)$ et stocke en mémoire les erreurs TD correspondantes (*voir* lignes 6–11). L'étape d'amélioration met à jour l'ensemble des paramètres θ, ν, et σ dans la direction du gradient estimé à partir de la moyenne empirique sur la base $\mathcal{D}$ en exploitant la séparabilité des approximations compatibles (*voir* lignes 12–16). Les taux d'apprentissage α_h^θ, α_h^ν et α_h^σ évoluent selon les conditions standard de Robbins et Monro pour les algorithmes d'approximation stochastiques [22], *c.à.d.* $\sum_{h=0}^\infty \alpha_h = \infty$, $\sum_{h=0}^\infty \alpha_h^2 < \infty$. De plus, si l'on suit les recommandations de [15], ces taux doivent être mis à jour à chaque itération tel que les paramètres des acteurs θ soient modifiés « plus lentement » que les paramètres ν and σ pour garantir la convergence. Pour faciliter la convergence d'une politique jointe pour une modification constante de ses paramètres, une méthode de choix est l'utilisation gradient naturel [1, 14]. L'algorithme ACDC naturel (NACDC) ne diffère de sa version initiale que par la formule de mise à jour des acteurs :

$$\theta_{t,h+1}^i \leftarrow \theta_{t,h}^i + \alpha_h^\theta \mathbb{E}_{\mathcal{D}_{t,h}}[\frac{a_t(u_t|o_t)}{\bar{a}_t(u_t|o_t)} \nu_{t,h}^i]$$

1. Il est néanmoins possible de le faire tourner de façon distribuée en donnant le moyen aux agents de collaborer pendant l'entraînement en se communiquant leur informations locales.

Algorithm 1: Actor-Critic for Decentralized Control (ACDC).

```
1  ACDC()
2      Initialize θ_0, ν_0, σ_0 arbitrarily and h ← 0.
3      while θ_h has not converged do
4          evaluation() and improvement()
5          h ← h + 1
6  evaluation()
7      Initialize D^h_{0:T} ← ∅
8      for j = 1...m and t = 0...T do
9          Create trajectories (x_{t:t+1}, o_{t:t+1}, u_t) ~ p(·|θ^{1:n}_{0:t})
10         Evaluate δ_t ← r_t + γV^σ_{t+1}(x_{t+1}, o_{t+1}) − V^σ_t(x_t, o_t)
11         Compose batch
           D_{t,h} ← {(o_t, u_t, δ_t, a_t(u_t|o_t)/ā_t(u_t|o_t))} ∪ D_{t,h}
12 improvement()
13     for i = 1...n do
14         Baseline update
           σ^i_{t,h+1} ← σ^i_{t,h} + α^σ_h E_{D_{t,h}}{δ_t (a_t(u_t|o_t)/ā_t(u_t|o_t)) φ^i_t(o^i_t)}
15         Critic update
           ν^i_{t,h+1} ← ν^i_{t,h} + α^ν_h E_{D_{t,h}}{δ_t (a_t(u_t|o_t)/ā_t(u_t|o_t)) φ^i_t(o^i_t, u^i_t)}
16         Actor update
           θ^i_{t,h+1} ← θ^i_{t,h} + α^θ_h E_{D_{t,h}}{(a_t(u_t|o_t)/ā_t(u_t|o_t)) φ^i_t(o^i_t, u^i_t) ν^i_{t,h}}
```

Pour conclure, cette section, quelques remarques sur les propriétés théoriques des algorithmes ACDC. D'une part, moyennant quelques conditions mineures, ils convergent avec probabilité 1 vers un optimum local puisque ce sont de véritables algorithmes d'ascension du gradient [6]. L'argument repose sur le fait qu'ils minimisent l'erreur de projection quadratique par descente de gradient stochastique, mais nous renvoyons le lecteur vers [6] pour plus de détails. D'autre part, ils terminent sur un optimum local qui est aussi un équilibre de Nash. En effet les dérivées partielles du critique centralisé par rapport à n'importe lequel de ses paramètres ne sont nulles qu'en un point d'équilibre, qui est aussi un optimum local.

6 Expérimentations

Dans cette section, nous montrons empiriquement les avantages du paradigme CTDC sur les paradigmes classiques CTCC et DTDC. Nos résultats semblent indiquer que les méthodes ACDC se comparent favorablement aux algorithmes existants sur de nombreux domaines multi-agents décentralisés de la littérature. Nous illustrons également les limitations de l'implémentation actuelle qui l'empêchent d'obtenir de meilleures performances.

6.1 Conditions expérimentales

Comme nous l'avons laisser transparaître tout au long de cet article, plusieurs facteurs clés peuvent affecter les performances des méthodes acteur-critique. Parmi ceux-ci, nous relèverons : le paradigme d'entraînement utilisé (CTCC *vs* DTDC *vs* CTDC) ; la représentation de la poli-

tique (stationnaire *vs* non-stationnaire) ; les structures d'approximation (linéaires *vs* réseaux de neurones récurrents (RNN)) ; la représentation des historiques (troncatures à mémoire finie *vs* états cachés d'un RNN). Nous implémentons trois variantes des méthodes acteur-critique qui combinent ces différents facteurs. Sauf mention contraire, nous ferons référence à chaque variante mise en œuvre par le nom du paradigme qu'elle implémente, par exemple CTDC pour notre algorithme `ACDC`, suivi de la structure de représentation interne utilisée quand cela est pertinent, par exemple : « *CTDC_trunc(*K*)* » pour un algorithme `ACDC` utilisant un historique tronqué des K dernières observations comme entrée d'une politique linéaire non-stationnaire, ou encore « *DTDC_rnn* » pour l'algorithme `Reinforce` distribué utilisant un réseau de neurone récurrent dans une politique stationnaire (voir Figure 2).

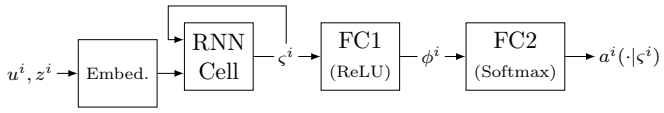


FIGURE 2 – Réseau de neurones récurrent utilisé comme structure « acteur » de chaque agent $i \in \mathcal{I}_n$. Une couche LSTM met à jour un état caché interne à partir du précédent et d'une encapsulation (embedding) d'une paire commande-observation. Elle est suivie d'un perceptron dont les activations sont rectifiées linéairement (ReLU) qui génère un vecteur de caractéristiques (features) ϕ^i, qui sont combinées linéairement par un second perceptron dont la sortie est normalisée par un softmax pour donner la règle de décision conditionnelle $a^i(\cdot|\varsigma^i)$.

Nous avons mené nos expérimentations sur une station de calcul *Dell Precision Tower 7910* équipé d'un CPU *Intel Xeon* à 16 cœurs cadencés à 3GHz, de 16Go de RAM et d'une carte graphique *nVIDIA Quadro K620* munie de 2Go de mémoire vidéo. Nous avons simulé plusieurs bancs de test standard de la littérature des Dec-POMDP, – *Dec. Tiger*, *Broadcast Channel*, *Mars*, *Box Pushing*, *Meeting in a Grid*, et *Recycling Robots* pour les citer – tel qu'ils sont défini sur `http://masplan.org`. Le détail des meta-paramètres utilisés est donné dans le Tableau 1 en annexe.

6.2 Importance de la représentation de l'état interne

Dans cette section, nous mettons en place des expériences visant à comprendre la façon dont la représentation des historiques affectent les performances des méthodes `ACDC`. La Figure 3 comparent les récompenses cumulées moyennes obtenues avec des historiques tronqués de taille 1 et 3, celles obtenues avec des RNN, et la performance ϵ-optimal donnée par l'algorithme de planification centralisé `FB-HSVI` [10]. Pour des horizons courts, ici $T = 10$, CTDC rnn converge rapidement vers de bonnes solutions par rapport à *CTDC_trunc(*1*)* et *CTDC_trunc(*3*)*. Ceci laisse penser que *CTDC_rnn* apprends des représentations des historiques plus concises et discriminantes que la représentation tronquée. Il semblerait néanmoins que cette différence de performances s'amenuise quand l'horizon de planification augmente (non illustré ici). On notera néanmoins une certaine instabilité de la récompense cumulée empirique pour les RNN sur les tâches plus complexe comme *Dec. Tiger*, un banc de test mettant en avant l'importance de la collecte d'information avant de prendre une action décisive, où les pénalités en cas d'erreur sont, en valeur absolues, bien plus élevées que les récompenses en cas de succès.

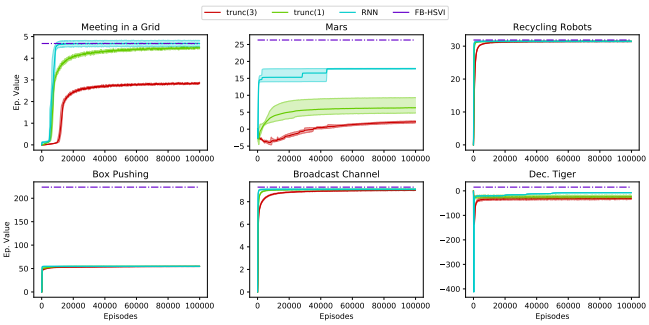


FIGURE 3 – Comparaison des différentes structures de représentation des historiques à $T = 10$.

Dans l'ensemble, nos expériences montrent un fort potentiel des RNN pour représenter les historiques. Ils ont l'avantage par rapport aux historiques tronqués d'apprendre automatiquement des classes d'équivalence et une représentation compacte de l'état interne, en se basant seulement sur la retro-propagation du gradient issu du signal de récompense. Il est aussi important de noter ici l'importance du choix des méta-paramètres, en particulier le taux d'apprentissage, qui, s'il est trop élevé, encourage des convergences prématurées vers des optima locaux insatisfaisants, sans laisser le temps à la politique d'explorer le reste des trajectoires possibles. Attention également à certaines propriétés spécifiques des modèles considérés, pour lesquelles les méthodes par ascension de gradient peinent à échapper au piège d'optima locaux. Nous n'avons pas pu identifier avec certitude quelles caractéristiques particulières avaient le plus d'impact négatif sur les performances, et nous laissons à de futures études l'analyse et l'exploration de méthodes plus fiables pour entraîner ces architectures.

6.3 Comparaison des différents paradigmes

Dans cette section, nous comparons les paradigmes CTCC, DTDC et CTDC en utilisant indifféremment les représentations ayant données les meilleurs résultats dans chaque cas. Nous incluons également à la comparaison les résultats issus de deux autres algorithmes de la littérature Dec-POMDP : un algorithme de planification ϵ-optimal appelé `FB-HSVI` [10], et un algorithme de planification par échantillonnage appelé Espérance-Maximisation Monte-Carlo (`MCEM`) [30] ; un choix justifié par de nombreuses similarités avec les méthodes acteur critique. Nous souli-

gnons le fait que nous ne cherchons pas à égaler les performances de FB−HSVI qui est un algorithme de planification centralisé nécessitant la connaissance à priori du modèle. Il nous fournit des performances de référence très proches de l'optimal global. Quant à MCEM, les résultats sont repris de [30] [2].

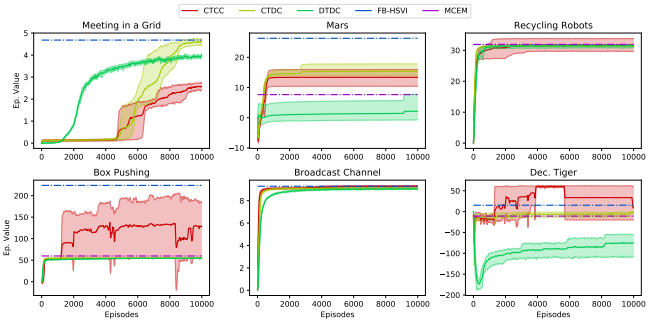


FIGURE 4 – Comparaison des paradigmes pour $T = 10$.

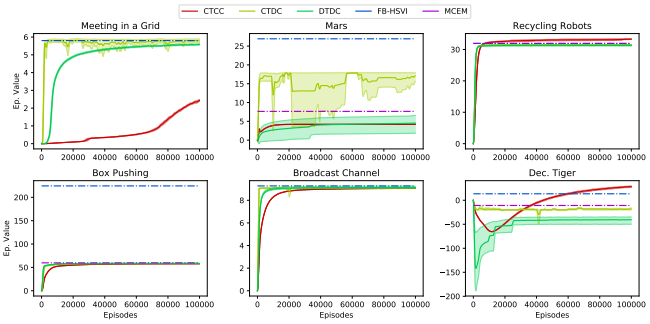


FIGURE 5 – Comparaison des paradigmes pour $T = \infty$.

Pour presque tous les bancs de test, CTDC semble prendre le meilleur sur les deux autres paradigmes, que ce soit à horizon fini ou infini. CTCC pâtit probablement du fléau de la dimension dans l'espace des historiques joints, et ne parvient pas à l'explorer de façon suffisamment efficace avant que les taux d'apprentissage rendent les mises à jour des paramètres négligeables, ou que le nombre maximum d'itérations fixé au départ ne soit atteint. Le fait d'utiliser la politique apprise comme politique d'échantillonnage (on-policy) amplifie très certainement cet effet. Ayant un espace d'historiques individuels bien plus réduit à explorer, CTDC donne de meilleur résultats que CTCC dans ces expérimentations. Comparé à DTDC qui explore lui-aussi des espaces d'historiques de dimensions plus raisonnables, le paradigme CTDC a un net avantage, et l'utilisation d'un critique centralisé compatible donnent de meilleures performances sur la vitesse de convergence et la qualité de l'optimum local atteint. Bien que CTDC donnent des résultats favorables – ou au moins comparables – par rapport à l'algorithme de l'état de l'art MCEM, il y a encore une large marge d'amélioration pour atteindre les optima globaux donnés par FB−HSVI pour tous les bancs de test. Comme mentionné précédemment, c'est en partie dû à une compression encore imparfaite des historiques, mais aussi à des limitations intrinsèques des méthodes par ascension de gradient, qui ne peuvent garantir qu'un optimum local.

2. Sauf pour *Meeting in a Grid* et *Broadcast Channel* pour lesquels les valeurs indiquées dans [30] étaient bien au delà de l'optimal, laissant penser à une anomalie.

7 Conclusion

Cet article pose les fondations théoriques des méthodes acteur-critique pour les Dec-POMDP dans le paradigme CTDC. Dans ce paradigme, un algorithme acteur-critique centralisé apprend des politiques indépendantes, une par agent, guidé par un unique critique joint. Nous montrons qu'un critique centralisé compatible peut s'écrire comme la somme de critiques individuels, qui sont chacun des combinaisons linéaires des « caractéristiques » de la politique individuelle correspondante. Nos expérimentations montrent que nos méthodes acteur-critique appelées ACDC se démarquent favorablement des approches standard du RL pour un certain nombre de bancs de test de la littérature. L'implémentation actuelle de ACDC soulève deux problèmes bien connu du domaine : le compromis exploitation-exploration et la représentation des états internes individuels. En particulier pour ce dernier point, apprendre à projeter les historiques individuels vers des états internes proches des états d'occupation individuels est un défi de taille, auquel nous comptons continuer à contribuer dans le futur. Outre ce problème de représentation des historiques individuels, ACDC peut exploiter la séparabilité du critique joint compatible pour passer à l'échelle d'un assez grand nombre d'agents. Nous nous intéressons à présent à une application multi-agents décentralisée à large échelle, pour laquelle nous cherchons à exploiter cette propriété.

Références

[1] Shun-Ichi AMARI. "Natural Gradient Works Efficiently in Learning". In : *Neural Comput.* 10.2 (1998). ISSN : 0899-7667.

[2] Karl J ASTRÖM. "Optimal Control of Markov Decision Processes with Incomplete State Estimation". In : *Journal of Mathematical Analysis and Applications* 10 (1965).

[3] Richard E BELLMAN. "The Theory of Dynamic Programming". In : *Bulletin of the American Mathematical Society* 60.6 (1954).

[4] Daniel S BERNSTEIN et al. "The Complexity of Decentralized Control of Markov Decision Processes". In : *Mathematics of Operations Research* 27.4 (2002).

[5] Craig BOUTILIER. "Planning, Learning and Coordination in Multiagent Decision Processes". In : *Proc. of the Sixth Conf. on Theoretical Aspects of Rationality and Knowledge*. 1996.

[6] Thomas DEGRIS, Martha WHITE et Richard S. SUTTON. “Linear Off-Policy Actor-Critic”. In : *Proc. of the 29th Int. Conf. on ML, ICML 2012, Edinburgh, Scotland, UK*. 2012.

[7] Jilles Steeve DIBANGOYE et al. “Exploiting Separability in Multi-Agent Planning with Continuous-State MDPs”. In : *Proc. of the Thirteenth Int. Conf. on Autonomous Agents and Multiagent Systems*. 2014.

[8] Jilles Steeve DIBANGOYE et al. “Optimally Solving Dec-POMDPs As Continuous-state MDPs”. In : *Proc. of the Twenty-Fourth Int. Joint Conf. on AI*. 2013.

[9] Jilles Steeve DIBANGOYE et al. *Optimally solving Dec-POMDPs as Continuous-State MDPs : Theory and Algorithms*. Research Report RR-8517. 2014.

[10] Jilles S DIBANGOYE et al. “Optimally Solving Dec-POMDPs as Continuous-State MDPs”. In : *Journal of AI Research* 55 (2016).

[11] Jakob N. FOERSTER et al. “Counterfactual Multi-Agent Policy Gradients”. In : *CoRR* (2017).

[12] Jayesh K. GUPTA, Maxim EGOROV et Mykel KOCHENDERFER. “Cooperative Multi-agent Control Using Deep Reinforcement Learning”. In : *Autonomous Agents and Multiagent Systems*. 2017.

[13] Eric A HANSEN, Daniel S BERNSTEIN et Shlomo ZILBERSTEIN. “Dynamic Programming for Partially Observable Stochastic Games”. In : *Proc. of the Nineteenth National Conf. on AI*. 2004.

[14] Sham KAKADE. “A Natural Policy Gradient”. In : *Advances in Neural Information Processing Systems 14 (NIPS 2001)*. 2001.

[15] Vijay R. KONDA et John N. TSITSIKLIS. “Actor-Critic Algorithms”. In : *Advances in Neural Information Processing Systems 12*. 2000.

[16] Landon KRAEMER et Bikramjit BANERJEE. “Multi-agent reinforcement learning as a rehearsal for decentralized planning”. In : *Neurocomputing* 190 (2016).

[17] Ryan LOWE et al. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In : *Advances in Neural Information Processing Systems 30*. 2017.

[18] Volodymyr MNIH et al. “Human-level control through deep reinforcement learning”. In : *Nature* 518.7540 (fév. 2015). ISSN : 0028-0836.

[19] Matej MORAVČÍK et al. “DeepStack : Expert-level artificial intelligence in heads-up no-limit poker”. In : *Science* 356.6337 (2017).

[20] Frans A OLIEHOEK et al. “Incremental Clustering and Expansion for Faster Optimal Planning in Dec-POMDPs”. In : *Journal of AI Research* 46 (2013).

[21] Leonid PESHKIN et al. “Learning to Cooperate via Policy Search”. In : *Sixteenth Conf. on Uncertainty in Artificial Intelligence (UAI-2000)*. 2000.

[22] H ROBBINS et S MONRO. “A stochastic approximation method”. In : *The annals of mathematical statistics* 22.3 (1951).

[23] Yoav SHOHAM et Kevin LEYTON-BROWN. *Multiagent Systems : Algorithmic, Game-Theoretic, and Logical Foundations*. New York, NY, USA, 2008. ISBN : 0521899435.

[24] Richard S SUTTON et Andrew G BARTO. *Introduction to Reinforcement Learning*. 2nd. Cambridge, MA, USA, 2016. ISBN : 0262193981.

[25] Richard S SUTTON et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In : *Proc. of the 12th Int. Conf. on Neural Information Processing Systems*. Cambridge, MA, USA, 1999.

[26] Daniel SZER et François CHARPILLET. “An Optimal Best-First Search Algorithm for Solving Infinite Horizon DEC-POMDPs”. In : *Proc. of the Fifteenth European Conf. on ML*. 2005.

[27] Daniel SZER, François CHARPILLET et Shlomo ZILBERSTEIN. “MAA* : A Heuristic Search Algorithm for Solving Decentralized POMDPs”. In : *Proc. of the Twenty-First Conf. on Uncertainty in AI*. 2005.

[28] Ming TAN. “Multi-agent Reinforcement Learning : Independent vs. Cooperative Agents”. In : *Readings in Agents*. San Francisco, CA, USA, 1998.

[29] Ronald J WILLIAMS. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In : *ML* 8.3 (1992).

[30] Feng WU, Shlomo ZILBERSTEIN et Nicholas R JENNINGS. “Monte-Carlo Expectation Maximization for Decentralized POMDPs”. In : *Proc. of the Twenty-Fourth Int. Joint Conf. on AI*. 2013.

[31] Xinhua ZHANG, Douglas ABERDEEN et S. V. N. VISHWANATHAN. “Conditional Random Fields for Multi-agent Reinforcement Learning”. In : *Proc. of the 24th international conference on Machine learning* (2007).

A Meta-paramètres pour ACDC

Représentation	RNN	trunc(K)
Horizon	10 / ∞^1	10 / ∞^1
Escompte	1 / 0.9	1 / 0.9
Nombre de tests	3	3
Itérations	100000	100000
Mini-batch	256	32
Taux d'apprentissage initial α_0	0.05	1.0
Taux d'apprentissage minimal $\alpha_{\min}$	0	0
Amortissement du taux λ_α	0.96	0.37
Pas d'amortissement κ_α	40000	80000
Type d'erreur	Monte-Carlo	Time-Difference
Saturation du gradient	20	Aucune
Échantillonage	On-policy	On-policy

TABLE 1 – Meta-paramètres pour les méthodes ACDC. - Note[1] : Les algorithmes ont échantillonné des trajectoires de taille T telle que $\frac{1}{(1-\gamma^T)} \max_{x,u} R(x,u) \ll \epsilon$

Formule de mise à jour du taux d'apprentissage :

$$\alpha_h = \alpha_{\min} + (\alpha_0 - \alpha_{\min}) \times \exp\left(\frac{h}{\lambda_\alpha} \log \kappa_\alpha\right)$$

B Preuves

B.1 Preuve du Lemme 2

Démonstration.

$$\begin{aligned}
\frac{\partial f(x)}{\partial x^j} &= f(x)\frac{\partial \log f(x)}{\partial x^j} \text{ (dérivée de la composition } \log \circ f) \\
&= f(x)\frac{\partial}{\partial x^j}\left(\log \prod_{i=1}^{n} f^i(x^i)\right) \text{ (séparabilité)} \\
&= f(x)\frac{\partial}{\partial x^j}\sum_{i=1}^{n} \log f^i(x^i) \text{ (propriétés du log)} \\
&= f(x)\sum_{i=1}^{n}\frac{\partial \log f^i(x^i)}{\partial x^j} \\
&= f(x)\frac{\partial \log f^j(x^j)}{\partial x^j} \text{ (termes nuls } \forall i \neq j)
\end{aligned}$$

□

B.2 Preuve du Lemme 3

Démonstration. Soit $\mathcal{X}$ le domaine de la variable aléatoire X.

$$\begin{aligned}
\mathbb{E}_{X\sim p}\left[f(X)\right] &= \int_{x\in\mathcal{X}} p(x)f(x)dx \\
&= \int_{x\in\mathcal{X}} \frac{q(x)}{q(x)}p(x)f(x)dx \\
&= \int_{x\in\mathcal{X}} q(x)\frac{p(x)}{q(x)}f(x)dx \\
&= \mathbb{E}_{X\sim q}\left[\frac{p(X)}{q(X)}f(X)\right]
\end{aligned}$$

□

B.3 Preuve du Lemme 4

Démonstration. Calculons la dérivée partielle de $V_t^\pi(x_t, o_t)$ par rapport à θ_t^i :

$$\frac{\partial V_t^\pi(x_t,o_t)}{\partial \theta_t^i} \doteq \frac{\partial}{\partial \theta_t^i}\sum_{u_t\in U} a_t(u_t|o_t)Q_t^\pi(x_t,o_t,u_t) \tag{23}$$

$$= \sum_{u_t\in U}\Big[\frac{\partial a_t(u_t|o_t)}{\partial \theta_t^i}Q_t^\pi(x_t,o_t,u_t) + a_t(u_t|o_t)\frac{\partial Q_t^\pi(x_t,o_t,u_t)}{\partial \theta_t^i}\Big] \tag{24}$$

$$= \sum_{u_t\in U}\left[\frac{\partial a_t(u_t|o_t)}{\partial \theta_t^i}Q_t^\pi(x_t,o_t,u_t) + 0\right] \tag{25}$$

Dans l'équation 25, on utilise le fait que $\frac{\partial Q_t^\pi(x_t,o_t,u_t)}{\partial \theta_t^i} = 0$. En effet, si l'on se réfère à l'équation de Bellman du Lemme 1, ni la récompense immédiate, ni l'espérance sur la Q-valeur à $t+1$ ne dépend de θ_t^i. La preuve se conclut aisément en appliquant le Lemme 2 sur $a_t(u_t|o_t)$, puis en introduisant $\bar{a}$ par le Lemme 3. □

B.4 Preuve du Théorème 1

Cas à horizon fini.

Démonstration. Soit $t = 1, \ldots, T$. Séparons d'abord la mesure de performance $J(s_0; \theta_{0:T}^{1:n})$ en deux termes : une récompense passée du temps 0 au temps $t-1$ et une récompense future de t à T.

$$\begin{aligned}
J(s_0;\theta_{0:T}^{1:n}) &\doteq \mathbb{E}_{(X_0,O_0)\sim\mathbb{P}(\cdot|M_n)}[V_0^\pi(X_0,O_0)] \\
&= \mathbb{E}_{(x_0,o_0)\sim\mathbb{P}(\cdot|M_n)}[\mathbb{E}_{u_0\sim a_0(\cdot|o_0)}[Q_0^\pi(x_0,o_0,u_0)]] \\
&= \mathbb{E}_{(X_0,O_0,U_0)\sim\mathbb{P}(\cdot|a_0,M_n)}[Q_0^\pi(X_0,O_0,U_0)] \\
&= \mathbb{E}_{(X_{0:1},O_{0:1},U_{0:1})\sim\mathbb{P}(\cdot|a_{0:1},M_n)}[R(X_0,U_0) + \gamma Q_1^\pi(X_1,O_1,U_1)] \\
&= \mathbb{E}_{(X_{0:t},O_{0:t},U_{0:t})\sim\mathbb{P}(\cdot|a_{0:t},M_n)}[R(\omega_{0:t-1}) + \gamma^t Q_t^\pi(X_t,O_t,U_t)] \\
&= J(s_0;\theta_{0:t-1}^{1:n}) + \gamma^t J(s_t;\theta_{t:T}^{1:n})
\end{aligned}$$

où $J(s_0;\theta_{0:t-1}^{1:n})$ dénote la mesure de performance selon la politique jointe partielle $a_{0:t-1}$ en partant de l'état d'occupation initial s_0 :

$$J(s_0;\theta_{0:t-1}^{1:n}) \doteq \mathbb{E}_{(X_{0:t-1},O_{0:t-1},U_{0:t-1})\sim\mathbb{P}(\cdot|a_{0:t-1},M_n)}[R(\omega_{0:t-1})]$$

et $J(s_t;\theta_{t:T}^{1:n})$ dénote la mesure de performance selon la politique jointe partielle $a_{t:T}$ en partant de l'état d'occupation s_t définit par $s_t(X_t,O_t) \doteq \mathbb{P}(X_t,O_t|a_{0:t-1},M_n)$:

$$\begin{aligned}
J(s_t;\theta_{t:T}^{1:n}) &\doteq \mathbb{E}_{(X_{0:t},O_{0:t},U_{0:t-1})\sim\mathbb{P}(\cdot|s_t,M_n)}[V_t^\pi(X_t,O_t)] \\
&= \mathbb{E}_{(X_{0:t},O_{0:t},U_{0:t-1})\sim\mathbb{P}(\cdot|s_t,M_n)}[\mathbb{E}_{U_t\sim a_t(\cdot|O_t)}[Q_t^\pi(X_t,O_t,U_t)]] \\
&= \mathbb{E}_{(X_{0:t},O_{0:t},U_{0:t})\sim\mathbb{P}(\cdot|s_t,a_t,M_n)}[Q_t^\pi(X_t,O_t,U_t)]
\end{aligned}$$

Calculons ensuite les dérivées partielles de $J(s_0;\theta_{0:t-1}^{1:n})$ et $\gamma^t J(s_t;\theta_{t:T}^{1:n})$ par rapport à θ_t^i pour n'importe quel $i \in I_n$:

$$\frac{\partial J(s_1;\theta_{0:T}^{1:n})}{\partial \theta_t^i} = \gamma^t \mathbb{E}_{(X_{0:t},O_{0:t},U_{0:t-1})\sim\mathbb{P}(\cdot|s_t,M_n)}\left[\frac{\partial V_t^\pi(X_t,O_t)}{\partial \theta_t^i}\right] \tag{26}$$

en remarquant que $\partial J(s_0;\theta^{1:n}_{0:t-1})/\partial\theta^i_t = 0$. On applique ensuite le Lemme 4 à partir de (26) pour conclure la preuve. □

Cas à horizon infini.

Démonstration. Développons d'abord l'expression de la dérivée partielle de la fonction de valeur : $\forall x \in \mathcal{X}, \forall \varsigma$,

$$\begin{aligned}
\frac{\partial V^\pi(x,\varsigma)}{\partial\theta^i} &= \frac{\partial}{\partial\theta^i}\left(\mathbb{E}_{U\sim a(\cdot|\varsigma)}\left[Q^\pi(x,\varsigma,U)\right]\right)\\
&= \frac{\partial}{\partial\theta^i}\left(\sum_{u\in\mathcal{U}} a(u|\varsigma)Q^\pi(x,\varsigma,u)\right)\\
&= \sum_{u\in\mathcal{U}}\Big(\frac{\partial a(u|\varsigma)}{\partial\theta^i}Q^\pi(x,\varsigma,u) + a(u|\varsigma)\frac{\partial Q^\pi(x,\varsigma,u)}{\partial\theta^i}\Big)\\
&= \sum_{u\in\mathcal{U}} a(u|\varsigma)\left[\frac{\partial\log a^i(u^i|\varsigma^i)}{\partial\theta^i}Q^\pi(x,\varsigma,u) + \frac{\partial Q^\pi(x,\varsigma,u)}{\partial\theta^i}\right]\\
&= \mathbb{E}_{U\sim a(\cdot|\varsigma)}\left[\frac{\partial\log a^i(U^i|\varsigma^i)}{\partial\theta^i}Q^\pi(x,\varsigma,U) + \frac{\partial Q^\pi(x,\varsigma,U)}{\partial\theta^i}\right]
\end{aligned}$$

Détaillons à présent la dérivée partielle de $Q^\pi(x,\varsigma,u)$:

$$\begin{aligned}
\frac{\partial Q^\pi(x,\varsigma,u)}{\partial\theta^i} &= \frac{\partial}{\partial\theta^i}\left(R(x,u) + \gamma\mathbb{E}_{\substack{X',Z\sim p(\cdot|x,u)\\ \Sigma'\sim\psi(\cdot|\varsigma,u,Z)}}\left[V^\pi(X',\Sigma')\right]\right)\\
&= \gamma\mathbb{E}_{\substack{X',Z\sim p(\cdot|x,u)\\ \Sigma'\sim\psi(\cdot|\varsigma,u,Z)}}\left[\frac{\partial V^\pi(X',\Sigma')}{\partial\theta^i}\right]
\end{aligned}$$

On ré-injecte ce résultat :

$$\begin{aligned}
\frac{\partial V^\pi(x,\varsigma)}{\partial\theta^i} &= \mathbb{E}_{U\sim a(\cdot|\varsigma)}\left[\frac{\partial\log a^i(U^i|\varsigma^i)}{\partial\theta^i}Q^\pi(x,\varsigma,U)\right]\\
&\quad + \gamma\mathbb{E}_{\substack{U\sim a(\cdot|\varsigma)\\ X',Z\sim p(\cdot|x,U)\\ \Sigma'\sim\psi(\cdot|\varsigma,U,Z)}}\left[\frac{\partial V^\pi(X',\Sigma')}{\partial\theta^i}\right]
\end{aligned}$$

Répétons à nouveau ce développement avec $\frac{\partial V^\pi(x',\Sigma')}{\partial\theta^i}$ dans l'expression précédente pour faire apparaître un schéma récurrent :

$$\begin{aligned}
\frac{\partial V^\pi(x,\varsigma)}{\partial\theta^i} &= \mathbb{E}_{U\sim a(\cdot|\varsigma)}\left[\frac{\partial\log a^i(U^i|\varsigma^i)}{\partial\theta^i}Q^\pi(x,\varsigma,U)\right]\\
&+ \gamma\mathbb{E}_{\substack{U\sim a(\cdot|\varsigma)\\ X',Z\sim p(\cdot|x,U)\\ \Sigma'\sim\psi(\cdot|\varsigma,U,Z)\\ U'\sim a(\cdot|\Sigma')}}\left[\frac{\partial\log a^i(U'^{,i}|\Sigma'^{,i})}{\partial\theta^i}Q^\pi(X',\Sigma',U')\right]\\
&+ \gamma^2\mathbb{E}_{\substack{U\sim a(\cdot|\varsigma)\\ X',Z\sim p(\cdot|x,U)\\ \Sigma'\sim\psi(\cdot|\varsigma,U,Z)}}\left[\mathbb{E}_{\substack{U'\sim a(\cdot|\Sigma')\\ X'',Z'\sim p(\cdot|X',U')\\ \Sigma''\sim\psi(\cdot|\Sigma',U',Z')}}\left[\frac{\partial V^\pi(X'',\Sigma'')}{\partial\theta^i}\right]\right]
\end{aligned}$$

Et ainsi de suite jusqu'à obtenir :

$$\begin{aligned}
\frac{\partial V^\pi(x,\varsigma)}{\partial\theta^i} &= \lim_{t\to\infty}\sum_{k=0}^{t}\gamma^k\mathbb{E}_{U,X',\Sigma'\sim\mathbb{P}^k(\cdot|M_n,a,\psi,x,\varsigma)}\Big[\\
&\qquad \frac{\partial\log a^i(U^i|\Sigma'^{,i})}{\partial\theta^i}Q^\pi(X',\Sigma',U)\Big]
\end{aligned}$$

où $\mathbb{P}^k(x',\varsigma'|M_n,a,\psi,x_0,o_0)$ dénote la probabilité de rencontrer l'état x' et la représentation interne ς' après avoir suivi la politique a pendant k pas de temps, dans un environnement M_n en partant de l'état x et de la représentation interne ς. La formule finale s'obtient en prenant l'espérance de l'expression précédente sur la distribution initiale des états s_0. □

B.5 Preuve du Théorème 2

Démonstration. La preuve montre que l'estimation du gradient basée sur les approximations $V^\sigma_{0:T}$ et $A^\nu_{0:T}$, telles que $\sigma^{1:n}_{0:T}$ et $\nu^{1:n}_{0:T}$ satisfassent l'une des deux conditions, préserve la direction du véritable gradient de la politique. Nous dériverons cette preuve dans le cas où la seconde condition est vérifiée, car un raisonnement tout à fait similaire tient dans le premier cas. Dans ce second cas, $\sigma^{1:n}_{0:T}$ doit minimiser l'erreur quadratique moyenne (MSE) $\mathbb{E}[\epsilon^2_t(X_t,O_t,U_t)]$. Pour tout temps $t = 0,1,\ldots,T$, pour tout agent $i \in I_n$:

$$\frac{\partial\mathbb{E}[\epsilon^2_t(X_t,O_t,U_t)]}{\partial\nu^i_t} = 2\mathbb{E}[\epsilon_t(X_t,O_t,U_t)\frac{\partial A^\nu_t(X_t,O_t,U_t)}{\partial\nu^i_t}].$$

car la distribution de X_t, O_t, U_t ne dépend pas de ν. Comme $\nu^{1:n}_{0:T}$ satisfait la condition (18), on a :

$$\frac{\partial\mathbb{E}[\epsilon^2_t(X_t,O_t,U_t)]}{\partial\nu^i_t} = \mathbb{E}[\epsilon_t(X_t,O_t,U_t)\frac{\partial\log a^i_t(U^i_t|O^i_t)}{\partial\theta^i_t}].$$

En développant l'expression de $\epsilon_t(X_t,O_t,U_t)$, et en réarrangeant les différents termes, on obtient :

$$\begin{aligned}
&\mathbb{E}\left[Q^\pi_t(x_t,o_t,u_t)\frac{\partial\log a^i_t(u^i_t|o^i_t)}{\partial\theta^i_t}\right]\\
&= \mathbb{E}\left[\big(V^\sigma_t(x_t,o_t) + A^\nu_t(x_t,o_t,u_t)\big)\frac{\partial\log a^i_t(u^i_t|o^i_t)}{\partial\theta^i_t}\right]
\end{aligned}$$

Grâce au Lemme 4 l'expression devient :

$$\begin{aligned}
&\frac{\partial V^\pi_t(x_t,o_t)}{\partial\theta^i_t}\\
&= \mathbb{E}\left[\big(V^\sigma_t(x_t,o_t) + A^\nu_t(x_t,o_t,U_t)\big)\frac{\partial\log a^i_t(U^i_t|o^i_t)}{\partial\theta^i_t}\right]
\end{aligned}$$

Puis en utilisant le Lemme 3 pour introduire la politique d'exploration $\bar\pi$, on déduit l'expression final (19). □

Learning to Act in Continuous Dec-POMDPs

Jilles S. Dibangoye[1] Olivier Buffet[2]

[1] INSA de Lyon, laboratoire CITI, INRIA
[2] LORIA, INRIA

prénom.nom@inria.fr

Résumé

Nous nous attaquons au problème d'apprentissage par renforcement dans le cadre des processus décisionnels de Markov partiellement observables et décentralisés. Les tentatives précédentes ont conduit à différentes variantes de la méthode généralisée d'itération de politiques, qui dans le meilleur des cas abouties à des optima locaux. Dans ce papier, nous nous restreindrons au plans, qui sont des formes plus simples que des politiques. Nous dériverons, sous certaines conditions, le premier algorithm algorithme optimal d'apprentissage par renforcement coopératif. Afin d'accroître le passage à l'échelle de cet algorithme, nous remplacerons l'opérateur glouton traditionnel par un programme linéaire en nombre entier. Les résultats expérimentaux montrent que notre méthode est capable d'apprendre de façon optimale dans plusieurs bancs de test de la litérature.

Mots Clef

Processes décisionnels de de Markov partiallement observables et décentralisés, Apprentissage par Renforcement.

Abstract

We address a long-standing open problem of reinforcement learning in continuous decentralized partially observable Markov decision processes. Previous attempts focussed on different forms of generalized policy iteration, which at best led to local optima. In this paper, we restrict attention to plans, which are simpler to store and update than policies. We derive, under mild conditions, the first optimal cooperative multi-agent reinforcement learning algorithm. To achieve significant scalability gains, we replace the greedy maximization by mixed-integer linear programming. Experiments show our approach can learn to act optimally in many finite domains from the literature.

Keywords

Decentralized Markov Decision Partially Observable Processes, Reinforcement Learning.

1 Introduction

Decentralized partially observable Markov decision processes (Dec-POMDPs) emerged as the standard framework for sequential decision making by a team of collaborative agents Bernstein et al. [2000]. A key assumption of Dec-POMDPs is that agents can neither see the actual state of the system nor explicitly communicate their noisy observations with each other due to communication cost, latency or noise, hence providing a partial explanation of the double exponential growth at every control interval of the required memory in optimal algorithms Hansen et al. [2004], Szer et al. [2005], Oliehoek et al. [2008, 2013], Dibangoye et al. [2016]. While planning methods for finite Dec-POMDPs made substantial progress in recent years, the formal treatment of the corresponding reinforcement learning problems received little attention so far, let alone its continuous counterpart. The literature of multi-agent reinforcement learning (MARL) can be divided into two main categories: *concurrent* and *team approaches* Tan [1998], Panait and Luke [2005].

Perhaps the dominant paradigm in MARL is the concurrent approach, which involves multiple simultaneous learners: typically, each agent has its learning process. Self-interested learners, for example, determine their best-response behaviors considering their opponents are part of the environment, often resulting in local optima Brown [1951], Hu and Wellman [1998], Littman [1994]. While concurrent learning can apply in Dec-POMDPs, a local optimum may lead to severely suboptimal performances Peshkin et al. [2000], Zhang and Lesser [2011], Kraemer and Banerjee [2016]. Also, methods of this family face two conceptual issues that limit their applicability. The primary concern is that of the co-adaptation dilemma, which arises when each attempt to modify an agent behavior can ruin learned behaviors of its teammates. Another major problem is that of the multi-agent credit assignment, that is, how to split the collective reward signal among independent learners.

Alternatively, the team approach involves a single learner acting on behalf of all agents to discover a collective solution Salustowicz et al. [1998], Miconi [2003]. Interestingly, this approach circumvents the difficulties arising from both the co-adaptation and the multi-agent credit assignment.

Coordinated agents, for example, simultaneously learn their control choices and the other agent strategies assuming instantaneous and free explicit communications Guestrin et al. [2002], Kok and Vlassis [2004]. While methods of this family inherit from standard single-agent techniques, they need to circumvent two significant drawbacks: the explosion in the state space size; and the centralization of all learning resources in a single place. Recently, team algorithms ranging from Q-learning to policy-search have been introduced for finite Dec-POMDPs, but with no guaranteed global optimality Kraemer and Banerjee [2016], Wu et al. [2013], Liu et al. [2015, 2016]. So, it seems one can either compute local optima with arbitrary bad performances or calculate optimal solutions but assuming noise-free, instantaneous and explicit communications.

A recent approach to optimally solving finite Dec-POMDPs suggests recasting them into occupancy-state MDPs (oMDPs) and then applying (PO)MDP solution methods Dibangoye et al. [2016]. In these oMDPs, the states called occupancy states are distributions over hidden states and joint histories of the original problem, and actions called decision rules are mappings from joint histories to controls Nayyar et al. [2011], Oliehoek [2013], Dibangoye et al. [2016]. This approach achieves scalability gains by exploiting the piece-wise linearity and convexity of the optimal value function. Since this methodology successfully applied for planning in finite Dec-POMDPs, it is natural to wonder which benefits it could bring to the corresponding MARL problem. Unfortunately, a straightforward application of standard RL methods to oMDPs will face three severe limitations. First, occupancy states are unknown, and hence must be estimated. Second, they lie in a continuum making tabular RL methods inapplicable. Finally, the greedy maximization is computationally demanding in decentralized stochastic control problems Radner [1962], Kumar and Zilberstein [2009].

This paper extends the methodology of Dibangoye et al. to MARL, focussing on the three major issues that limit its applicability. Our primary result is the proof that, by restricting attention to plans instead of policies, a linear function over occupancy states and decision rules, which is simple to store and update, can capture the optimal performance for Dec-POMDPs. We further use plans instead of policies in a policy iteration algorithm, with the plan always being improved with respect to a linear function and a linear function always being driven toward the linear function for the plan. Under accurate estimation of the occupancy states, the resulting algorithm, called occupancy-state SARSA (oSARSA) Rummery, G. A. and Niranjan [1994], is guaranteed to converge with probability one to an optimal plan for any finite Dec-POMDP. To extend its applicability to higher-dimensional domains, oSARSA replaces the greedy (or soft) maximization by a mixed-integer linear program for finite settings, and a gradient approach for continuous ones. Altogether, we obtain a MARL algorithm that can apply to continuous Dec-POMDPs. Experiments show our approach can learn to act optimally in many finite domains from the literature.

We organize the remainder of this paper as follows. Section 2 extends a recent planning theory from finite to continuous settings, starting with a formal definition of finite and continuous Dec-POMDPs. We proceed with the introduction of a framework for centralized MARL in Dec-POMDPs in Section 3. Also, we discuss our solutions to the three limitations mentioned above. We present the resulting algorithm oSARSA along with convergence guarantees in Section 4. Finally, we conduct experiments in Section 5, demonstrating our approach can learn to act optimally in many finite domains from the literature.

2 Planning in Dec-POMDPs as oMDPs

2.1 Continuous Dec-POMDPs

A continuous Dec-POMDP is a tuple $M \doteq (n, X, \{U^i\}, \{Z^i\}, p, r, \ell, \gamma, b_0)$, where n denotes the number of agents involved in the decentralized stochastic control process; X is a set of hidden world states, denoted x or y; U^i is a private control set of agent $i \in [\![1; n]\!]$, where $U = U^1 \times \cdots \times U^n$ specifies the set of controls $u = (u^1, \ldots, u^n)$; Z^i is a private observation set of agent i, where $Z = Z^1 \times \cdots \times Z^n$ specifies the set of observations $z = (z^1, \ldots, z^n)$; p describes a transition probability kernel with conditional density $p^{u,z}(x, y)$; r is a reward model with immediate reward $r(x, u)$, we assume rewards are two-side bounded, i.e., for some $c \in \mathbb{R}^+$, $\forall x \in X, u \in U\colon |r(x, u)| \leqslant c$; ℓ is the planning horizon; $\gamma \in [0, 1]$ denotes the discount factor; and b_0 is the initial belief state with density $b_0(x_0)$. We shall restrict attention to finite planning horizon $\ell < \infty$ since an infinite planning horizon solution is within a small scalar $\epsilon > 0$ of a finite horizon solution where $\ell = \lceil \log_\gamma((1 - \gamma)\epsilon/c) \rceil$. A Dec-POMDP is usually either *finite*, i.e., sets X, U^i and Z^i for all $i \in [\![1; n]\!]$ are finite, or *continuous*, i.e., the same sets are continuous. The remainder of this section extends concepts and standard results from finite to continuous settings.

Because we are interested in MARL, we assume an incomplete knowledge about M, *i.e.,* p and r are either unavailable or only through a generative model. Hence, the goal of solving M is to find a joint plan, *i.e.,* a tuple of individual decision rules, one for each agent and time step: $\rho \doteq (a^1_{0:\ell}, \ldots, a^n_{0:\ell})$. A tth individual decision rule $a^i_t\colon O^i_t \mapsto \mathcal{P}(U^i)$ of agent i prescribes private controls based on the whole information available to the agent up to the tth time step, *i.e.,* history of controls and observations $o^i_t = (u^i_{0:t-1}, z^i_{1:t})$, where $o^i_0 = \varnothing$ and $o^i_t \in O^i_t$. A tth joint decision rule, denoted $a_t\colon O_t \mapsto \mathcal{P}(U)$, can be specified as $a_t(u|o) \doteq \prod_{i=1}^n a^i_t(u^i|o^i)$, where $O_t \doteq O^1_t \times \cdots \times O^n_t$, $o^i \in O^i_t$ and $o \doteq (o^1, \ldots, o^n) \in O_t$. From control interval t onward, agents collectively receive discounted cumulative rewards, denoted by random variable $R_t \doteq \gamma_1 r_t + \cdots + \gamma_\ell r_\ell$,

where α_t denotes the time-step dependent weighting factors, often set to $\gamma_t = \gamma^t$ for discounted problems or $\gamma_t = \frac{1}{\ell}$ for the average reward case. For any control interval t, joint plans $a_{0:t}$ of interest are those that achieve the highest performance measure $J(a_{0:t}) \doteq \mathbb{E}^{a_{0:t}}\{R_0 \mid b_0\}$ starting at b_0, where $\mathbb{E}^{a_{0:t}}\{\cdot\}$ denotes the expectation with respect to the probability distribution over state-action pairs joint plan $a_{0:t}$ induces, in particular $J(\rho) \doteq J(a_{0:\ell-1})$ for $\rho \doteq a_{0:\ell-1}$. One can show that, in Dec-POMDPs, there always exists a deterministic plan that is as good as any stochastic plan [see Puterman, 1994, Lemma 4.3.1]. Unfortunately, there is no direct way to apply the theory developed for Markov decision processes Bellman [1957], Puterman [1994] to Dec-POMDPs, including: the Bellman optimality equation; or the policy improvement theorem. To overcome these limitations, we rely on a recent theory by Dibangoye et al. that recasts M into an MDP, thereby allowing knowledge transfer from the MDP setting to Dec-POMDPs.

2.2 Occupancy-State MDPs

To overcome the fact that agents can neither see the actual state of the system nor explicitly communicate their noisy observations with each other, Szer et al. (2005) and later on Dibangoye et al. (2016) suggest formalizing M from the perspective of a centralized algorithm. A centralized algorithm acts on behalf of the agents by selecting a joint decision rule to be executed at each control interval based on all data available about the system, namely the information state. The information state at the end of control interval t, denoted $\iota_{t+1} \doteq (b_0, a_{0:t})$, is a sequence of joint decision rules the centralized algorithm selected starting at the initial belief state. Hence, the information state satisfies the following recursion: $\iota_0 \doteq (b_0)$ and $\iota_{t+1} \doteq (\iota_t, a_t)$ for all control interval t, resulting in an ever-growing sequence. To generalize the value from one information state to another one, Dibangoye et al. introduced the concept of occupancy states. The occupancy state at control interval t, denoted $s_t \doteq \mathbb{P}(x_t, o_t|\iota_t)$, is a distribution over hidden states and joint histories conditional on information state ι_t at control interval t. Interestingly, the occupancy state has many important properties. First, it is a sufficient statistic of the information state when estimating the (current and future) reward to be gained by executing a joint decision rule:

$$R(s_t, a_t) \doteq \int_X \int_O s_t(x, o) \int_U a_t(u|o) \cdot r(x, u) \mathtt{d}u \mathtt{d}o \mathtt{d}x.$$

In addition, it describes a deterministic and fully observable Markov decision process, where the next occupancy state depends only on the current occupancy state and next joint decision rule, for all $y \in X, o \in O, u \in U, z \in Z$:

$$T(s_t, a_t) \doteq s_{t+1}$$
$$s_{t+1}(y, (o, u, z)) \doteq a_t(u|o) \int_X s_t(x, o) \cdot p^{u,z}(x, y) \mathtt{d}x.$$

The process the occupancy states describe is known as the occupancy-state Markov decision process (*o*MDP), and denoted $M' \doteq (S, A, R, T, \ell, \gamma, s_0)$. This is an ℓ-steps deterministic and continuous MDP with respect to M, where $S \doteq \cup_{t \in [\![0;\ell-1]\!]} S_t$ is the set of occupancy states up to control interval $\ell - 1$; $A \doteq \cup_{t \in [\![0;\ell-1]\!]} A_t$ is the set of joint decision rules up to control interval $\ell - 1$; R is the reward model; and T is the transition rule; s_0 is the initial occupancy state, which is essentially the initial belief in M; γ and ℓ are as in M. It is worth noticing that there is no need to construct explicitly M'; instead we use M (when available) as a generative model for the occupancy states $T(s_t, a_t)$ and rewards $R(s_t, a_t)$, for all control intervals t.

Note that, using control terminology, planning (or learning) in M' can be made either open-loop or closed-loop. M is called an open-loop planning problem because the class of considered solutions (*i.e.,* plans) are only function of time (and not of the underlying occupancy states). It is called a closed-loop planning problem since it is a Markov decision process, whose solution is known to be a policy (mapping from occupancy states to joint decision rules). Policies $\pi \colon S \mapsto A$, mappings from occupancy states to decision rules, generalizes plans ρ, by prescribing a joint decision rule depending on the current time and occupancy state. Plans are in general sub-optimal compared to policies, as they prescribes a decision rule depending only upon the current time. However, here, both open- and closed-loop approaches can lead to an optimal solution. Below, we review a closed-loop approach based on the dynamic programming theory Bellman [1957].

Due to the deterministic nature of the dynamics in oMDP M', planning (or learning) in M' can be seen, without loss of optimality, not only as closed-loop—a solution being a policy $\pi \colon S \mapsto A$, mapping (encountered/reachable) occupancy states to joint decision rules—, but also as open-loop—a solution being a(n inconditional) plan ρ, that is, a sequence of joint decision rules. Below, we review a closed-loop approach based on the dynamic programming theory Bellman [1957].

For any finite M, the Bellman equation is written as follows: for all occupancy state $s_t \in S_t$, and some fixed policy π,

$$V_t^\pi(s_t) \doteq R(s_t, \pi(s_t)) + \gamma_1 V_{t+1}^\pi(T(s_t, \pi(s_t))) \quad (1)$$

with boundary condition $V_\ell^\pi(\cdot) \doteq 0$, describes the return of a particular occupancy state s_t when taking decision rule $a_t = \pi(s_t)$ prescribed by π. The equation for an optimal policy π^* is referred to as the Bellman optimality equation: for any control interval t, and occupancy state s_t,

$$V_t^*(s_t) \doteq \max_{a_t \in A} R(s_t, a_t) + \gamma_1 V_{t+1}^*(T(s_t, a_t)) \quad (2)$$

with boundary condition $V_\ell^*(\cdot) \doteq 0$. Unfortunately, occupancy states lie in a continuum, which makes exact dynamic programming methods infeasible. When optimized exactly, the value function solution of (2) along with the boundary condition is always piece-wise linear and convex in the occupancy-state space Dibangoye et al. [2016]. Similarly to continuous POMDPs Porta et al. [2006], we generalize this property to continuous Dec-POMDPs.

Lemma 1. *For any arbitrary M', the solution $V^*_{0:\ell}$ of (2) is convex in the occupancy-state space. If we restrict attention to deterministic policies and finite M (and corresponding M'), the solution of (2) is piece-wise linear and convex in the occupancy-state space. Hence, the optimal value at any occupancy state s_t is as follows:*

$$V^*_t(s_t) \doteq \max_{\alpha_t \in \Gamma_t} \langle s_t, \alpha_t \rangle, \tag{3}$$

where $\langle s_t, \alpha_t \rangle$ is used to express the expectation of a linear function α_t (also called α-function[1]) in the probability space defined by sample space $X \times O$, the σ-algebra $X \times O$ and the probability distribution s_t; and Γ_t is the set of all tth α-functions.

Proof. Consider the α-function induced when agents follow plan $a_{t:\ell-1}$ from control interval t onwards, denoted $\alpha^{a_{t:\ell-1}}$ and given by $\alpha^{a_{t:\ell-1}}(x, o) \doteq \mathbb{E}\{R_t | x_t = x, o_t = o, a_{t:\ell-1}\}$. Hence, the optimal value starting at any occupancy state s_t is given by taking the maximum over values of all possible plans from control interval t onwards: $V^*_t(s_t) = \max_{a_{t:\ell-1}} \langle s_t, \alpha^{a_{t:\ell-1}} \rangle$. In addition, the linearity of the expectation also implies that $\alpha^{a_{t:\ell-1}}$ is linear in the occupancy-state space. The proof directly follows from [Rockafellar, 1970, Theorem 5.5]. □

Lemma 1 shows that for any arbitrary M and corresponding M', the solution of (2), represented by sets $\Gamma_{0:\ell}$, is convex in the occupancy-state space. Each α-function defines the value function over a bounded region of the occupancy-state space. In addition, it is associated with a plan, defining the optimal plan for a bounded region of the occupancy-state space. Sets $\Gamma_{0:\ell}$ are iteratively improved by adding a new α-function that dominates current ones over certain regions of the occupancy-state space. The α-function to be added is computed using point-based Bellman backup operator $\mathbb{H}$:

$$[\mathbb{H}\Gamma_{t+1}](s_t) = \underset{\alpha^a_t \,:\, a \in A_t, \alpha_{t+1} \in \Gamma_{t+1}}{\arg\max} \langle s_t, \alpha^a_t \rangle,$$

where $\alpha^a_t(x, o) \doteq \mathbb{E}\{r(x, u) + \gamma_1 \alpha_{t+1}(y, (o, u, z)) | a\}$, for each hidden state $x \in X$, and joint history $o \in O$. Interestingly, $\mathbb{H}$ remains both a contracting and isotonic operator in continuous settings Porta et al. [2006]. To keep the number of α-functions manageable, one can prune those that are dominated over the entire occupancy-state space. All in all, the oMDP reformulation permits us to solve finite M by means of M' using near-optimal planning methods leveraging on the special structure of the optimal value function Shani et al. [2013]. This methodology results in the current state-of-the-art algorithm to optimally solving finite Dec-POMDPs Dibangoye et al. [2016]. So it seems natural to wonder if the same methodology can also succeed when applied to the corresponding reinforcement-learning problem. In other words, how can a centralized algorithm learn to coordinate a team of agents with possibly contradicting perceptual information?

[1] In continuous M, α-functions represent linear functions (including parametric ones); whereas in finite M, α-functions become α-vectors, that is finite-dimensional vectors.

3 Learning in Dec-POMDPs as oMDPs

Using the oMDP reformulation, a natural approach to achieve centralized RL for decentralized stochastic control suggests applying exact RL methods. In the Q-learning algorithm Watkins and Dayan [1992], for example, one would learn directly the Q-value function when following a fixed policy π: for any control interval $t \in [\![0; \ell - 1]\!]$,

$$Q^\pi_t(s_t, a_t) \doteq R(s_t, a_t) + \gamma_1 V^\pi_{t+1}(T(s_t, a_t)) \tag{4}$$

with boundary condition $Q^\pi_\ell(\cdot, \cdot) = 0$. The policy improvement theorem provides a procedure to change a sub-optimal policy π into an improved one $\bar{\pi}$ Howard [1960]: for any control interval $t \in [\![0; \ell - 1]\!]$,

$$\bar{\pi}(s_t) \doteq \arg\max_{a_t \in A_t} Q^\pi_t(s_t, a_t). \tag{5}$$

Unfortunately, this approach has three severe limitations. First, the occupancy states are unknown and must be estimated. Second, even if we assume a complete knowledge of the occupancy states, they lie in a continuum, which precludes exact RL methods to accurately predict α-functions even in the limit of infinite time and data. Finally, the greedy maximization required to improve the value function proved to be NP-hard in finite settings and problematic in continuous ones Radner [1962], Kumar and Zilberstein [2009].

3.1 Addressing Estimation Issues

Although mappings T and R in M' are unknown to either agents or a centralized algorithm, one can instead estimate on the fly both $T(s_0, a_{0:t-1})$ and $R(T(s_0, a_{0:t-1}), a_t)$ for some fixed plan $\rho \doteq a_{0:\ell-1}$ through successive interactions of agents with the environment. To this end, we shall distinguish between two settings. The first one assumes a generative model is available during the centralized learning phase, e.g. a black box simulator; and the second does not. In both cases, we build on the concept of replay pool Mnih et al. [2015], except that we extend it from stationary single-agent domains to non-stationary multi-agent domains.

If a generative model is available during the learning phase, then a Monte Carlo method can approximate $T(s_0, a_{0:t-1})$ and $R(T(s_0, a_{0:t-1}), a_t)$ arbitrary closely. To this end, the generative model allows the agents to sample experiences generated from M. A ℓ-steps experience is a 4-tuple $\xi \doteq (x_{0:\ell-1}, u_{0:\ell-1}, r_{0:\ell-1}, z_{1:\ell})$, where $x_{0:\ell-1}$ are sampled hidden states, $u_{0:\ell-1}$ are controls made, $r_{0:\ell-1}$ are reward signals drawn from the reward model, and $z_{1:\ell}$ are the resulting observations, drawn from the dynamics model. If we let $\mathcal{D}^\rho \doteq \{\xi^{[i]}\}_{i \in [\![1:K]\!]}$ be the replay pool of K i.i.d random samples created through successive interactions with the generative model, then empirical occupancy state $\hat{s}_t \approx T(s_0, a_{0:t-1})$ and reward $\hat{R}_t \approx R(T(s_0, a_{0:t-1}), a_t)$ corresponding to the current $\mathcal{D}^\rho$ are given by: for any control interval $t \in [\![0 : \ell - 1]\!]$,

$$\hat{s}_t(x, o) \doteq \tfrac{1}{K} \textstyle\sum_{i=1}^K \delta_x(x^{[i]}_t) \cdot \delta_o(u^{[i]}_{0:t}, z^{[i]}_{1:t}) \tag{6}$$

$$\text{and} \quad \hat{R}_t \doteq \tfrac{1}{K} \textstyle\sum_{i=1}^K r^{[i]}_t, \tag{7}$$

where $\delta_x(\cdot)$ and $\delta_o(\cdot)$ denote the delta-Dirac mass located in hidden state and joint history pair, respectively. By the law of large numbers the sequence of averages of these estimates converges to their expected values, and the standard-deviation of its error falls as $1/\sqrt{K}$ [Sutton and Barto, 1998, chapter 5]. The error introduced by Monte Carlo when estimating $T(\hat{s}_{t-1}, a_{t-1})$ instead of $T(s_0, a_{0:t-1})$ is upper bounded by $2\ell/\sqrt{K}$. The proof follows from the performance guarantee of the policy-search algorithm by Bagnell et al. [2004]. Hence, to ensure the learned value function is within $\epsilon > 0$ of the optimal one, one should set the replay-pool size to $K = O(4\frac{\ell^2}{\epsilon^2})$.

When no generative model is available, the best we can do is to store samples agents collected during the learning phase into replay pools $\mathcal{D}^\rho$, one experience for each episode within the limit size of K. We maintain only the K recent experiences, and may discard[2] hidden states since they are unnecessary for the updates of future replay pools and the performance measure. The rationale behind this approach is that it achieves the same performances as a Monte Carlo method for the task of approximating $T(s_0, a_{0:t-1})$ and $R(T(s_0, a_{0:t-1}), a_t)$ given a fixed plan $\rho \doteq a_{0:\ell-1}$. In fact, if we let $\mathcal{D}^\rho$ be a replay pool of K i.i.d. samples generated according to ρ, the empirical occupancy state $\hat{s}_t \approx T(s_0, a_{0:t-1})$ and reward $\hat{R}_t \approx R(T(s_0, a_{0:t-1}), a_t)$ corresponding to $\mathcal{D}^\rho$ are given by (6) and (7), respectively. One can further show this approach preserves performance guarantees similar to those obtained when using a generative model.

3.2 Addressing Prediction Issues

The key issue with large (possibly continuous) spaces of occupancy states and decision rules is that of generalization, that is, how experiences with a limited subset of occupancy states and decision rules can produce a good approximation over a much larger space. Fortunately, a fundamental property of *o*MDPs is the convexity of the optimal value function over the occupancy-state space, see Lemma 1. Building on this property, we demonstrate a simple yet important preliminary result before stating the main result of this section.

Lemma 2. *For any arbitrary M' (resp. M), the optimal Q-value function is the upper enveloppe of sets $\Omega^*_{0:\ell}$ of α-functions over occupancy states and joint decision rules: for any control interval t, $Q^*_t(s_t, a_t) = \max_{q_t \in \Omega^*_t} \langle s_\tau \odot a_\tau, q_t \rangle$, where $q_t \in \Omega^*_t$ are appropriate α-functions, and $s_\tau \odot a_\tau$ denotes the Hadamard product*[3].

Proof. We proceed by induction to prove this property. In the following we assume that all operations (e.g. integrals) are well-defined in the corresponding spaces. For control interval $t = \ell - 1$, we only have to take into account the immediate reward and, thus, we have that $Q^*_{\ell-1}(s_{\ell-1}, a_{\ell-1}) = R(s_{\ell-1}, a_{\ell-1})$. Therefore, if we define the set $\Omega^*_{\ell-1} = \{q_{\ell-1}\}$, where $q_{\ell-1}(x, o, u) \doteq r(x, u)$, the property holds at control interval $t = \ell - 1$. We now assume the property holds for control interval $\tau + 1$ and we show that it also holds for control interval τ. Using (2) and (4), we have that, $Q^*_\tau(s_\tau, a_\tau) = R(s_\tau, a_\tau) + \gamma_1 \max_{a_{\tau+1}} Q^*_{\tau+1}(T(s_\tau, a_\tau), a_{\tau+1})$, and by the induction hypothesis, let $s_{\tau+1} \doteq T(s_\tau, a_\tau)$:

$$Q^*_{\tau+1}(s_{\tau+1}, a_{\tau+1}) = \max_{q \in \Omega^*_{\tau+1}} \int_{X,O,U} s_\tau(x, o) a_\tau(u|o) \int_{X,Z,U} p^{u,z}(x, y) a_{\tau+1}(u'|o, u, z) q(y, (o, u, z), u') \mathtt{d}u \mathtt{d}o \mathtt{d}x \mathtt{d}y \mathtt{d}z \mathtt{d}u'.$$

With the above,

$$Q^*_\tau(s_\tau, a_\tau) = \max_{a \in A_{\tau+1}, q \in \Omega^*_{\tau+1}} \int_{X,O,U} s_\tau(x, o) a_\tau(u|o) \Big[r(x, u) + \gamma_1 \int_{Y,Z,U} p^{u,z}(x, y) a(u'|o, u, z) q(y, (o, u, z), u') \mathtt{d}y \mathtt{d}z \mathtt{d}u' \Big] \mathtt{d}u \mathtt{d}o \mathtt{d}x.$$

At this point, we can define the bracketed quantity as

$$q^{a_{\tau+1}}(x, o, u) \doteq r(x, u) + \gamma_1 \int_{X,Z,U} p^{u,z}(x, y) a_{\tau+1}(u'|o, u, z) q(y, (o, u, z), u') \mathtt{d}y \mathtt{d}z \mathtt{d}u'.$$

Note that α-function $q^{a_{\tau+1}}$ is independent of occupancy state s_τ and decision rule a_τ for which we are computing Q^*_τ. With this, we have that $Q^*_\tau(s_\tau, a_\tau) = \max_{q^a \colon a \in A_{\tau+1}, q \in \Omega^*_{\tau+1}} \langle s_\tau \odot a_\tau, q^a \rangle$ and, thus the lemma holds. □

Lemma 2 generalizes the convexity property demonstrated in Lemma 1 from optimal value functions over occupancy states to optimal value functions over occupancy states and decision rules. As a consequence, finite sets $\Omega^*_{0:\ell-1}$ of α-functions can produce solutions arbitrarily close to the optimal Q-value function $Q^*_{0:\ell-1}$. Though Q-value function $Q^*_{0:\ell-1}$ generalizes from a pair of occupancy state and decision rule to another one, storing and updating a convex hull is non trivial. Instead of learning the optimal Q-value function over all occupancy states and decision rules, we explore a simpler yet tractable alternative, which will prove sufficient to preserve ability to eventually find an optimal plan starting at initial occupancy state s_0.

Theorem 1. *For any arbitrary M' (resp. M), the Q-value function $Q^{\rho^*}_{0:\ell-1}$ under an optimal plan $\rho^* \doteq a^*_{0:\ell-1}$ starting at initial occupancy state s_0 is linear in occupancy states and decision rules: $Q^{\rho^*}_t(s_t, a_t) = \langle s_t \odot a_t, q^{\rho^*}_t \rangle$ where $q^{\rho^*}_t \doteq \arg\max_{q_t \in \Omega^*_t} \langle T(s_0, a^*_{0:t-1}) \odot a^*_t, q_t \rangle$.*

Proof. The proof derives directly from Lemma 2. First, notice that any arbitrary non-dominated joint plan ρ induces a sequence of α-functions $q^\rho_{0:\ell-1}$ stored in $\Omega^*_{0:\ell-1}$, which proves the Q-value function under a fixed plan is linear over occupancy states and joint decision rules. In addition, each α-function $q^\rho_t \in \Omega^*_t$ describes the expected

[2] Note that one should keep hidden states when available since they often speed up the convergence.

[3] $\forall (x, o, u) \colon [s_\tau \odot a_\tau](x, o, u) \doteq s_\tau(x, o) \cdot a_\tau(u|o)$.

returns from $t \in [\![0; \ell-1]\!]$ onward, when agents follow non-dominated joint plan ρ. If we let ρ^* be a greedy joint plan with respect to $Q^*_{0:\ell-1}$, then $q^{\rho^*}_{0:\ell-1}$ is maximal along $\{T(s_0, a^*_{0:t-1})\}_{t\in[\![0;\ell-1]\!]}$. □

Theorem 1 proves that the Q-function for a given optimal joint plan achieves performance at the initial occupancy state s_0 as good as the Q-value function for an optimal joint policy. Standard policy iteration algorithms search for an optimal joint policy, which requires a finite set of α-functions to approximate V^*/Q^*, hence the resulting PWLC approximator is tight almost everywhere. Building upon Theorem 1, we search for an optimal ρ, which requires only a single α-function to approximate V^ρ/Q^ρ, thus the resulting linear approximator is loose everywhere except in the neighborhood of a few points. The former approach may require less iterations before convergence to an optimal joint policy, but the computational cost of each iteration shall increase with the number of α-functions maintained. The latter approach may require much more iterations, but all iteration shares the same computational cost.

3.3 Addressing Plan Improvement Issues

This section introduces a procedure to improve a plan starting with a sub-optimal one.

Suppose we have determined the value function $V^\rho_{0:\ell-1}$ for any arbitrary $\rho \doteq a_{0:\ell-1}$. For some control interval $t \in [\![0; \ell-1]\!]$, we would like to know whether or not we should change decision rules $a_{0:t}$ to choose $\bar{a}_{0:t} \neq a_{0:t}$. We know how good it is to follow the current plan from control interval t onward—that is V^ρ_t—but would it be better or worse to change to the new plan? One way to answer this question is to consider selecting $\bar{a}_{0:t}$ at control interval t and thereafter following decision rules $a_{t+1:\ell-1}$ of the existing ρ. The value of the resulting joint plan is given by $J(\bar{a}_{0:t-1}) + \gamma_1 V^\rho_{t+1}(T(s_0, \bar{a}_{0:t-1}))$. The key criterion is whether this quantity is greater or less than $J(\rho)$. Next, we state the plan improvement theorem for occupancy-state Markov decision processes.

Theorem 2. *Let $\rho \doteq a_{0:\ell-1}$ and $\bar{\rho} \doteq \bar{a}_{0:\ell-1}$ be any pair of plans and $J_{0:\ell}$ be a sequence of α-functions such that, for all t, $J_t(x_t, o_t) \doteq \mathbb{E}\{\alpha_0 r_0 + \ldots + \alpha_t r_t | b_0, x_t, o_t, a_{0:t-1}\}$. Let $\bar{s}_t \doteq T(s_0, \bar{a}_{0:t-1})$ and $s_t \doteq T(s_0, a_{0:t-1})$ be occupancy states at any control interval $t \in [\![0; \ell-1]\!]$ under $\bar{\rho}$ and ρ, respectively. Then, $\langle \bar{a}_{0:t*-1}, a_{t*:\ell-1}\rangle$ such that $t^* = \arg\max_{t\in[\![0;\ell-1]\!]} \langle \bar{s}_t - s_t, J_t - \gamma_1 V^\rho_t\rangle$ is as good as, or better than, ρ.*

Proof. The proof follows from the difference between the performance measure of $\rho \doteq a_{0:\ell-1}$ and $\bar{\rho} \doteq \bar{a}_{0:\ell-1}$. Let $\varsigma_t(\bar{\rho}, \rho)$ be the advantage of taking plan $\langle \bar{a}_{0:t-1}, a_{t:\ell-1}\rangle$ instead of ρ: for any control interval $t \in [\![0; \ell-1]\!]$,

$$\begin{aligned}\varsigma_t(\bar{\rho}, \rho) &= J(\bar{a}_{0:t-1}) + \gamma_1 V^\rho_t(T(s_0, \bar{a}_{0:t-1})) - J(\rho)\\ &= J(\bar{a}_{0:t-1}) - J(a_{0:t-1}) + \gamma_1(V^\rho_t(\bar{s}_t) - V^\rho_t(s_t))\\ &= \langle \bar{s}_t - s_t, J_t - \gamma_1 V^\rho_t\rangle.\end{aligned}$$

If we let $t^* \doteq \arg\max_{t=0,1,\ldots,\ell-1} \varsigma_t(\bar{\rho}, \rho)$, then plan $\langle \bar{a}_{0:t*-1}, a_{t*:\ell-1}\rangle$ achieves the highest advantage among plan set $\{\langle \bar{a}_{0:t-1}, a_{t:\ell-1}\rangle\}_{t\in[\![0;\ell-1]\!]}$ constructed based on $\bar{\rho}$. If $t^* = 0$, then $\langle \bar{a}_{0:t*-1}, a_{t*:\ell-1}\rangle = \rho$, and no improved plans were found from plan set generated from $\bar{\rho}$. Otherwise, new $\langle \bar{a}_{0:t*-1}, a_{t*:\ell-1}\rangle$ must be better than ρ. □

The plan improvement theorem shows how, given $\rho \doteq a_{0:\ell-1}$ and α-function $q^\rho_{0:\ell-1}$, we can easily evaluate a change in ρ at any control interval to a particular (possibly improved) plan. To ease exploration towards promising plans, we investigate the ϵ-greedy maximization (or soft-maximization). At each control interval t and occupancy state s_t, it randomly selects $\hat{a}_t$ with probability ϵ; otherwise, it greedily selects $\hat{a}_t$ w.r.t. the current Q-value function Q^ρ_t:

$$\hat{a}_t \doteq \arg\max_{a_t:\ a^1_t\in A^1_t,\ldots,a^n_t\in A^n_t} Q^\rho_t(s_t, a_t),$$

where $\hat{\rho} \doteq \hat{a}_{0:\ell-1}$. Unfortunately, this operation proved to be NP-hard for finite M and problematic in continuous M Radner [1962], Kumar and Zilberstein [2009]. We present a mixed-integer linear programming method, which successfully performs the greedy maximization for finite M. Mixed-Integer Linear Program 1 builds on MacDermed and Isbell [2013], which introduced an integer program for the greedy maximization in finite M. We also exploit the occupancy state estimation, in which $\hat{s}_t$ replaces s_t, and the current α-function q^ρ_t.

Mixed-Integer Linear Program 1 (For finite M)**.**

$$\underset{a_t, a^1_t,\ldots,a^n_t}{\textit{Maximize}} \ \sum_x \sum_o \hat{s}_t(x,o) \sum_u a_t(u|o)\cdot q^\rho_t(o,u) \tag{8}$$

$$\textit{s.t.:}\ \sum_{u^j} a_t(u^j, u^i|o) = a^i_t(u^i|o^i),\ \forall i, u^i, o \tag{9}$$

$$\sum_u a_t(u|o) = 1,\ \forall o \tag{10}$$

where $\{a_t(u|o)\}$ and $\{a^i_t(u^i|o^i)\}$ are positive and boolean variables, respectively.

Mixed-Integer Linear program 1 optimizes positive variables $\{a_t(u|o)\}_{u\in U, o\in O_t}$, one positive variable for each control-history pair. More precisely, each variable represents the probability $a_t(u|o)$ of control u being taken given that agents experienced joint history o. Constraints must be imposed on these variables to ensure they form proper probability distributions (10), and that they result from the product of independent probability distributions (9), one independent probability distribution for each agent. In order to make the description of the conditional independence,

$$a_t(u|o) = a^1_t(u^1|o^1) \times \cdots \times a^n_t(u^n|o^n), \tag{11}$$

we use additional variables $\{a^i_t(u^i|o^i)\}_{i\in[\![1;n]\!], u^i\in U^i, o^i\in O^i_t}$. Marginalizing out both sides of (11) over all control-history pairs of all agents except agent i, denoted $-i$, leads to (9). That is not sufficient to ensure conditional independence in general. If we further constrain $\{a^i_t(u^i|o^i)\}$ to be boolean, then system of equations (9) implies (11).

Given (9) and (10), agent variables $\{a_t^i(u^i|o^i)\}_{u^i \in U^i, o^i \in O_t^i}$ describe a proper probability distribution, so we omit corresponding constraints. Our greedy maximization approach is fundamentally different from previous ones, including the integer program by MacDermed and Isbell [2013] and the constraint optimization program by Kumar and Zilberstein [2009], Dibangoye et al. [2016]. First, while previous approaches made use of boolean variables, we use both positive and boolean variables instead. Next, prior approaches optimize a value function represented as a convex hull; we optimize an α-function instead.

4 The oSARSA Algorithm

This section presents the oSARSA algorithm with tabular representations and function approximations (using either linear functions or deep neural networks) along with convergence guarantees. oSARSA algorithms are specializations of Policy Iteration, except that we use plans instead of policies. For the sake of conciseness, we describe a generic algorithm, which can fit to either tabular or approximate representations.

In Dec-POMDPs, the goal of oSARSA is to learn $q^*_{0:\ell-1}$, a sequence of α-vectors of an optimal plan ρ^*. In particular, we must estimate $q_t(x, o, u)$ for the current plan ρ and for all reachable state x, joint history o, control u, and any control interval t. At the same time, the algorithm changes ρ towards improved plans according to the plan improvement theorem. The improved plans are constructed by exploring the occupancy-state space according to ϵ-greedy plans (see Section 3.3). To provide good estimations, we store all experiences in data set $\mathcal{D}^\rho$, from which we estimate the occupancy states and returns under ρ for any control interval (see Section 3.1). Upon estimating occupancy state $\hat{s}$ and selecting joint decision rule a, we update parametrized α-function q_t with parameter θ_t using q_{t+1}, $\mathcal{D}^\rho$ and a_{t+1} by means of temporal difference learning: for all (x, o, u),

$$\theta_t^{[\tau+1]} \doteq \theta_t^{[\tau]} + \beta_\tau \mathbb{E}_{\hat{s},a,\mathcal{D},a_{t+1}}\{\delta_t \nabla q_t^{[\tau]}(x, o, u)\} \quad (12)$$
$$\delta_t = r + \gamma_1 q_{t+1}^{[\tau]}(y, o', u') - q_t^{[\tau]}(x, o, u),$$

where β_τ is a step size, and quantity $\nabla q_t(x, o, u)$ denotes the gradient of q_t at (x, o, u) w.r.t. some parameter θ_t. Using tabular representations (e.g., finite/small M), $\theta_t = q_t$ and thus $\nabla q_t(x, o, u)$ is a unit vector $e_{x,o,u}$ whose value at (x, o, u) is one and zero otherwise. Using linear function approximations (e.g., continuous/large M), $q_t(x, o, u) \doteq \phi_t(x, o, u)^\top \theta_t$, where $\nabla q_t(x, o, u) = \phi_t(x, o, u)$ is the feature vector at (x, o, u). Algorithm 1 shows the pseudocode of oSARSA.

To establish the convergence of oSARSA, we introduce the following assumptions.

Theorem 3. *Consider assumptions: (1) The stepsizes $\{\beta_\tau\}_{\tau=1,2,\dots}$ satisfy Robbins and Monro's conditions; (2) The occupancy states $\hat{s}_{0:\ell-1}$ and immediate returns $\hat{R}_{0:\ell-1}$ are accurately estimated; and (3) Every pair of reachable occupancy state and joint decision rule is visited infinitely often. Under these assumptions, the sequence $q_{0:\ell-1}^{[\tau]}$ generated by oSARSA converges with probability 1 to $q^*_{0:\ell-1}$.*

Algorithm 1 The oSARSA Algorithm

Initialize $\bar{g} = -\infty$, $\bar{\rho}$ and $q_{0:\ell-1}$ arbitrary, and $\mathcal{D}^{\bar{\rho}}$.
while $q_{0:\ell-1}$ has not converged **do**
  Select ϵ-greedily ρ w.r.t. $q_{0:\ell-1}$ and $\mathcal{D}^{\bar{\rho}}$.
  Compose $\mathcal{D}^\rho$ with N trajectories $\{\xi^{[\tau]}\}_{\tau=1}^N$.
  Estimate (g, ς) from $[\sum_{t=0}^{\ell-1} \hat{R}_t | \mathcal{D}^\rho, \hat{s}_0 = s_0]$.
  If $g - \varsigma \geqslant \bar{g}$ **then** $(\bar{\rho}, \bar{g}, \mathcal{D}^{\bar{\rho}}) = (\rho, g + \varsigma, \mathcal{D}^\rho)$.
  Update α-functions $q_{0:\ell-1}$ as described in (12).
end while

Proof. Under these assumptions, we define $\mathbb{H}^\rho$ that maps a sequence of α-vectors $q_{0:\ell-1}$ to a new sequence of α-vectors $\mathbb{H}^\rho q_{0:\ell-1}$ according to the formula: for all hidden state x, joint history o and control u, at control interval t,

$$(\mathbb{H}^\rho q_{0:\ell-1})(x, o, u) = r(x, u) + \gamma_1 \mathbb{E}\{v_{t+1}(y, o \oplus (u, z))\},$$

where $v_t(x, o) \doteq q_t(x, o, \rho(o))$ and $\rho(o)$ is the control prescribed by ρ at joint history o. Then, the plan evaluation step of the oSARSA algorithm is of the form

$$q_t^{[\tau+1]}(x, o, u) = (1 - \beta_t) q_t^{[\tau]}(x, o, u) + \beta_t \kappa_t^{[\tau]}(x, o, u),$$
$$\kappa_t^{[\tau]}(x, o, u) = (\mathbb{H}^\rho q_{0:\ell-1}^{[\tau]})(x, o, u) + w_t(x, o, u),$$

where $w_t(x, o, u) = r(x, u) + \gamma_1 v_{t+1}^{[\tau]}(y, o \oplus (u, z)) - (\mathbb{H}^\rho q_{0:\ell-1}^{[\tau]})(x, o, u)$ is a zero mean noise term. Using this temporal-difference update-rule, see (12), we converge with probability 1 to $q^\rho_{0:\ell-1}$. It now remains to be verified that the plan improvement step of the oSARSA algorithm changes the current plan for an improved one. Initially, $\bar{g}$ is arbitrarily bad, so any new plan is an improved one. Then, $\bar{g} = J(\rho)$ for the current best plan ρ since occupancy state and return are accurately estimated. Hence, when ever $g \geqslant \bar{g}$, we know that the new plan $\bar{\rho}$ yields a performance measure $J(\bar{\rho})$ superior to $J(\rho)$, thus $\bar{\rho}$ improves ρ. We conclude the proof noticing that in finite M, the number of deterministic plans is finite. As a consequence, by visiting infinitely often every pair of occupancy state and decision rule we are guaranteed to visit all deterministic plans, hence an optimal one. □

It is now important to observe that we meet assumption (2) in Theorem 3 only when M is available. Otherwise, we rely on confidence bounds $[g - \varsigma, g + \varsigma]$, e.g. *Hoeffding's inequality*, on estimate $g \approx J(\rho)$. In particular, we use lower-bounds $g - \varsigma$ on sample means instead of the sample means g themselves, to limit situations where g is overestimated. Small data sets often lead to suboptimal solutions, but as the number of experiences in data set $\mathcal{D}^\rho$ increases, sample means and corresponding lower bounds get close to the mean, i.e., ς tends to 0. It is worth noticing that the memory complexity of the oSARSA algorithms is linear

with the size of an α-function, *i.e.,* $\mathcal{O}(|\mathcal{D}^\rho|)$; and its time complexity is linear with the episodes.

5 Experiments

We ran the oSARSA algorithm on a Mac OSX machine with 3.8GHz Core i5 and 8GB of available RAM. We solved the MILPs using ILOG CPLEX Optimization Studio. We define features to use sequences of K last joint observations instead of joint histories, hence the dimension of the parameter vector θ is $|X|(|U||Z|)^K$ for finite M.

We evaluate our algorithm on multiple benchmarks from the literature all available at `masplan.org`: Mabc, Recycling, Gridsmall, Grid3x3corners, Boxpushing, and Tiger. These are the largest and the most challenging benchmarks from the Dec-POMDP literature. For each of them, we compare our algorithm to the state-of-the-art algorithms based on either a complete or a generative model: FB-HSVI Dibangoye et al. [2016], RLaR Kraemer and Banerjee [2016], and MCEM Wu et al. [2013]. We also reported results of the state-of-the-art *model-free* solver: (distributed) REINFORCE Peshkin et al. [2000]. For REINFORCE and oSARSA, we used hyper-parameters ϵ and β ranging from 1 to 10^{-3} with a decaying factor of 10^4, sample size $|\mathcal{D}| = 10^4$. We use maximum episodes and time limit 10^5 and 5 hours, respectively, as our stopping criteria.

Surprisingly, REINFORCE performs very well on domains that consist of weakly coupled agents, *see* Figure 1. However, for domains with strongly coupled agents, *e.g.,* Tiger or BoxPushing, it often gets stuck at some local optima. In contrast, oSARSA converges to near-optimal solutions when enough ressources are available over all domains, *see* Figure 1 and Table 1. Regarding the most challenging benchmarks, which require more resources, oSARSA stops before the convergence to a near-optimal solution; yet, it often outperforms the other RL algorithms. RLaR can achieve near-optimal result for small domains and short planning horizon ($\ell \leqslant 5$), assuming there exists a unique optimal plan. As for MCEM, it can solve infinite horizon problems, but similarly to REINFORCE may get stuck in local optima; this is essentially as they both use a form of gradient descent in a parametrized policy space.

T	RLaR	MCEM	REINFORCE	**oSARSA**	FB-HSVI
	Tiger ($\|X\| = 2, \|Z\| = 4, \|U\| = 9, K = 3$)				
3	5.19	N.A.	5.0	5.19	5.19
4	4.46	N.A.	4.6	4.80	4.80
5	6.65	N.A.	2.2	6.99	7.02
6	–	N.A.	0.3	2.34	10.38
7	–	N.A.	-1.7	2.25	9.99
∞	N.A.	-10	-19.9	-0.2	13.44
	Grid3x3corners ($\|X\| = 81, \|Z\| = 81, \|U\| = 25, K = 1$)				
6	–	N.A.	1.46	1.49	1.49
7	–	N.A.	2.17	2.19	2.19
8	–	N.A.	2.96	2.95	2.96
9	–	N.A.	3.80	3.80	3.80
10	–	N.A.	4.66	4.69	4.68
	Boxpushing ($\|X\| = 100, \|Z\| = 16, \|U\| = 25, K = 1$)				
3	66.08	N.A.	17.6	65.27	66.08
4	98.59	N.A.	18.1	98.16	98.59
5	–	N.A.	35.2	107.64	107.72
6	–	N.A.	36.4	120.26	120.67
7	–	N.A.	36.4	155.21	156.42
8	–	N.A.	52.9	186.04	191.22
9	–	N.A.	54.5	206.75	210.27
10	–	N.A.	54.7	218.39	223.74
∞	N.A.	59.1	58.9	144.57	224.43

Table 1: Comparing $V^\rho(s_0)$ of all solvers when available, where the "–" sign mean "out of memory" and/or "out of time".

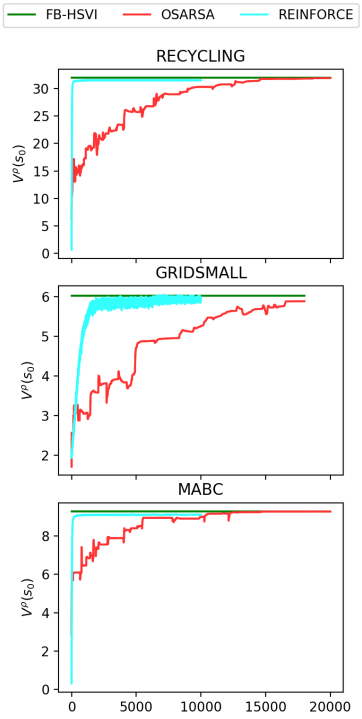


Figure 1: Comparing $V^\rho(s_0)$ of solvers with $\ell = \infty$ and $\gamma = 0.9$.

6 Discussion

This paper extends a recent but growing (deep) MARL paradigm Szer et al. [2005], Dibangoye et al. [2016], Kraemer and Banerjee [2016], Mordatch and Abbeel [2017], Foerster et al. [2017], namely RL for decentralized control, from model-based to model-free settings. This paradigm allows a centralized algorithm to learn on behalf of all agents how to select an optimal joint decision rule to be executed at each control interval based on all data available about the system during a learning phase, while still preserving ability for each agent to act based solely on its private histories at the execution phase. In particular, we introduced tabular and approximate oSARSA algorithms, which demonstrated promising results often outperforming state-of-the-art MARL approaches for Dec-POMDPs. To do so, oSARSA learns a value function that maps pairs of occupancy state and joint decision rule to reals. To ease the generalization in such high-dimensional continuous spaces, we restrict attention to plans rather than policies, which in turn restricts value functions of interest to linear functions. To speed up the greedy maximization, we used a MILP for finite settings—we shall use a gradient approach instead of a MILP for continuous settings in future works. Finally, we present a proof of optimality for a MARL algorithm when the estimation error is neglected. We shall investigate an approach to relax this somewhat restrictive assumption, perhaps within the probably approximately correct learning

framework.

The RL for decentralized control paradigm is significantly different from the standard RL paradigm, in which agents have the same amount of information during both the learning and the execution phases. Another major difference lies in the fact that learned value functions in standard (deep) RL algorithms are mapping from histories (or states) to reals. In contrast, oSARSA learns a value function that maps occupancy-state/decision-rule pairs to reals—spaces of occupancy states and joint decision rules are multiple orders of magnitude larger than history or state spaces. As a consequence, standard (MA)RL methods, *e.g.* REINFORCE and MCEM, may converge towards a local optimum faster than oSARSA, but the latter often converges towards a near-optimal solution. oSARSA uses occupancy states instead of joint histories mainly because occupancy states are (so far minimal) sufficient statistics for optimal decision-making in Dec-POMDPs—using joint histories instead of occupancy states may lead to suboptimal solutions except in quite restrictive settings. For example, RLaR learns value functions mapping history/action pairs to reals, but convergence towards an optimal solution is guaranteed only for domains that admit a unique optimal joint plan—which essentially restricts to POMDPs Kraemer and Banerjee [2016].

References

J. A. Bagnell, S. M. Kakade, J. G. Schneider, and A. Y. Ng. Policy Search by Dynamic Programming. In *Advances in Neural Information Processing Systems 16*. 2004.

R. E. Bellman. *Dynamic Programming*. 1957.

D. S. Bernstein, S. Zilberstein, and N. Immerman. The Complexity of Decentralized Control of Markov Decision Processes. In *Proc. of the Sixteenth Conf. on Uncertainty in AI*, 2000.

G. W. Brown. Iterative Solutions of Games by Fictitious Play. In *Activity Analysis of Production and Allocation*. 1951.

J. S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet. Optimally Solving Dec-POMDPs as Continuous-State MDPs. *Journal of AI Research*, 55, 2016.

J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual Multi-Agent Policy Gradients. *CoRR*, 2017.

C. Guestrin, M. G. Lagoudakis, and R. Parr. Coordinated Reinforcement Learning. In *Proc. of the Eighteenth Int. Conf. on ML*, San Francisco, CA, USA, 2002.

E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic Programming for Partially Observable Stochastic Games. In *Proc. of the Nineteenth National Conf. on AI*, 2004.

R. A. Howard. *Dynamic Programming and Markov Processes*. 1960.

J. Hu and M. P. Wellman. Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm. In *Proc. of the Fifteenth Int. Conf. on ML*, San Francisco, CA, USA, 1998.

J. R. Kok and N. Vlassis. Sparse Cooperative Q-learning. In *Proc. of the Twentieth Int. Conf. on ML*, New York, NY, USA, 2004.

L. Kraemer and B. Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

A. Kumar and S. Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proc. of the Eighth Int. Conf. on Autonomous Agents and Multiagent Systems*, 2009.

M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. of the Eleventh Int. Conf. on ML*, 1994.

M. Liu, C. Amato, X. Liao, L. Carin, and J. P. How. Stick-breaking policy learning in Dec-POMDPs. In *Int. Joint Conf. on AI (IJCAI) 2015*. AAAI, 2015.

M. Liu, C. Amato, E. P. Anesta, J. D. Griffith, and J. P. How. Learning for Decentralized Control of Multiagent Systems in Large, Partially-Observable Stochastic Environments. In *AAAI*, 2016.

L. C. MacDermed and C. Isbell. Point Based Value Iteration with Optimal Belief Compression for Dec-POMDPs. In *Advances in Neural Information Processing Systems 26*, 2013.

T. Miconi. When Evolving Populations is Better Than Coevolving Individuals: The Blind Mice Problem. In *Proc. of the 18th Int. Joint Conf. on AI*, IJCAI'03, 2003.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540), feb 2015. ISSN 0028-0836.

I. Mordatch and P. Abbeel. Emergence of Grounded Compositional Language in Multi-Agent Populations. *CoRR*, abs/1703.0, 2017.

A. Nayyar, A. Mahajan, and D. Teneketzis. Optimal Control Strategies in Delayed Sharing Information Structures. *Automatic Control, IEEE Transactions on*, 56(7), 2011.

F. A. Oliehoek. Sufficient Plan-Time Statistics for Decentralized POMDPs. In *Proc. of the Twenty-Fourth Int. Joint Conf. on AI*, 2013.

F. A. Oliehoek, M. T. J. Spaan, and N. A. Vlassis. Optimal and Approximate Q-value Functions for Decentralized POMDPs. *Journal of AI Research*, 32, 2008.

F. A. Oliehoek, M. T. J. Spaan, C. Amato, and S. Whiteson. Incremental Clustering and Expansion for Faster Optimal Planning in Dec-POMDPs. *Journal of AI Research*, 46, 2013.

L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3), 2005.

L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to Cooperate via Policy Search. In *Sixteenth Conf. on Uncertainty in Artificial Intelligence (UAI-2000)*, 2000.

J. M. Porta, N. Vlassis, M. T. J. Spaan, and P. Poupart. Point-Based Value Iteration for Continuous POMDPs. *J. Mach. Learn. Res.*, 7, 2006. ISSN 1532-4435.

M. L. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. Hoboken, New Jersey, 1994.

R. Radner. Team Decision Problems. *Ann. Math. Statist.*, 33(3), 1962.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, 22(3), 1951. ISSN 0003-4851.

R. T. Rockafellar. *Convex analysis*. Princeton Mathematical Series. Princeton, N. J., 1970.

M. Rummery, G. A. and Niranjan. On-line Q-learning using connectionist systems. Technical report, Cambridge University Engineering Department, 1994.

R. Salustowicz, M. Wiering, and J. Schmidhuber. Learning Team Strategies: Soccer Case Studies. *ML*, 33(2-3), 1998.

G. Shani, J. Pineau, and R. Kaplow. A survey of point-based POMDP solvers. *Journal of Autonomous Agents and Multi-Agent Systems*, 27(1), 2013.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *Proc. of the Twenty-First Conf. on Uncertainty in AI*, 2005.

M. Tan. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Readings in Agents*. San Francisco, CA, USA, 1998.

C. J. C. H. Watkins and P. Dayan. Q-Learning. *ML*, 8(3), 1992.

F. Wu, S. Zilberstein, and N. R. Jennings. Monte-Carlo Expectation Maximization for Decentralized POMDPs. In *Proc. of the Twenty-Fourth Int. Joint Conf. on AI*, 2013.

C. Zhang and V. Lesser. Coordinated Multi-Agent Reinforcement Learning in Networked Distributed POMDPs. In *Proc. of the Twenty-Fifth AAAI Conf. on AI*, San Francisco, California, USA, 2011.

Sur la Compilation de Jeux de Prédiction Combinatoire

Frédéric Koriche

CRIL, CNRS UMR 8188, Université d'Artois

frederic.koriche@cril.fr

Résumé

En optimisation en-ligne, le but est de choisir séquentiellement des solutions de manière à minimiser le coût moyen au cours du temps. Dès lors que l'espace de solutions faisables est décrit par des contraintes combinatoires, le problème est généralement NP-difficile. Dans cet article, nous investiguons l'idée de compiler un ensemble de contraintes combinatoires un circuit arithmétique de type `dDNNF`*, pour lequel les opérations d'optimisation linéaire et d'échantillonnage de solutions prennent un temps linéaire. Dans ce cadre de compilation, nous présentons des algorithmes efficaces pour diverses stratégies de prédiction en-ligne, en portant une attention particulière sur les approches de type « descente miroir ». Ces stratégies sont expérimentalement comparées sur plusieurs benchmarks du monde réel, pour lesquels les contraintes on été préalablement compilées en un circuit* `dDNNF`*.*

Mots Clef

Apprentissage en-ligne, compilation de connaissances, jeux de prédiction combinatoire.

Abstract

In online optimization, the goal is to iteratively choose solutions from a decision space, so as to minimize the average cost over time. As long as this decision space is described by combinatorial constraints, the problem is generally intractable. In this paper, we consider the paradigm of compiling the set of combinatorial constraints into a deterministic and Decomposable Negation Normal Form (`dDNNF`*) circuit, for which the tasks of linear optimization and solution sampling take linear time. Based on this framework, we provide efficient characterizations of existing combinatorial prediction strategies, with a particular attention to mirror descent techniques. These strategies are compared on several real-world benchmarks for which the set of Boolean constraints is preliminarily compiled into a* `dDNNF` *circuit.*

Keywords

Online learning, knowledge compilation, combinatorial prediction games.

1 Introduction

L'optimisation combinatoire est un domaine important de l'informatique et des mathématiques discrètes, avec un large spectre d'applications s'étendant depuis l'allocation de ressources et l'ordonnancement, jusqu'à la planification d'actions et les logiciels de configuration. Un des problèmes récurrents de ce domaine est de minimiser une fonction de perte linéaire $\boldsymbol{\ell}$ sur un espace combinatoire $\mathcal{S} \subseteq \{0,1\}^d$ de solutions faisables, représenté de manière compacte par un ensemble de contraintes. Dans la version *hors-ligne* de ce problème, toute l'information nécessaire à la spécification de la tâche d'optimisation est fournie à l'avance, et le but est de construire des algorithmes qui sont meilleurs que l'énumération naïve de toutes les solutions possibles. Dans la version *en-ligne* de ce problème, la fonction objectif $\boldsymbol{\ell}$ est susceptible de changer au cours du temps. Le défi est ici encore plus difficile, puisque l'algorithme d'optimisation doit choisir de manière séquentielle des solutions dans $\mathcal{S}$, afin de minimiser le coût moyen de ses choix.

De manière conceptuelle, le problème d'optimisation combinatoire en-ligne peut être vu comme un jeu répétitif entre un algorithme d'apprentissage et son environnement [1, 2]. Durant chaque tour t du jeu, l'apprenant choisit une solution faisable $\boldsymbol{s}_t$ dans l'espace $\mathcal{S}$ et, simultanément, l'environnement sélectionne un vecteur de perte $\boldsymbol{\ell}_t \in [0,1]^d$. A la fin du tour, l'apprenant subit la perte $\langle \boldsymbol{\ell}_t, \boldsymbol{s}_t \rangle = \sum_{i=1}^{d} \ell_t(i) s_t(i)$ et, selon le feedback fourni par son environnement, met à jour sa stratégie afin d'améliorer la chance de trouver de meilleures solutions durant les prochains tours.

Diverses classes de jeux de prédiction combinatoire peuvent être déclinées selon le type d'espace de décision et le type de feedback observé. Dans cet article, nous étudions les jeux à *information totale*, dans lesquels il est supposé que le feedback émis par l'environnement au tour t est le vecteur $\boldsymbol{\ell}_t$. En d'autres termes, l'apprenant connaît parfaitement la décision prise par l'environnement à la fin de chaque tour. En revanche, nous supposons que $\mathcal{S}$ peut être décrit par n'importe quelle formule `SAT`, c'est-à-dire un ensemble arbitraire de contraintes représentées par des clauses booléennes. Comme les codages `SAT` sont fréquemment utilisés dans les applications académiques et industrielles [5], notre cadre conceptuel couvre une classe importante de jeux de prédiction combinatoire.

En apprentissage en-ligne, la performance d'un algorithme de prédiction est caractérisée par deux mesures. La première est le *regret* : c'est la différence de perte cumulative entre l'algorithme de prédiction et la meilleure solution possible, qui aurait été choisie avec le recul sur l'historique du jeu. Dans cette étude, nous ne faisons aucune hypothèse sur la séquence des vecteurs de perte ; en particulier, $\boldsymbol{\ell}_t$ peut dépendre des décisions antérieures $\boldsymbol{s}_1, \cdots, \boldsymbol{s}_{t-1}$ choisies par l'apprenant. Pour ces environnements "sans oubli", pouvant être vus comme des adversaires, l'apprenant est autorisé à choisir ses décisions de manière aléatoire, et sa performance prédictive est mesurée par le *regret espéré* :

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} \langle \boldsymbol{\ell}_t, \boldsymbol{s}_t \rangle\right] - \min_{\boldsymbol{s} \in \mathcal{S}} \sum_{t=1}^{T} \langle \boldsymbol{\ell}_t, \boldsymbol{s} \rangle$$

La seconde mesure de performance est la *complexité calculatoire*, c'est-à-dire la quantité de ressources nécessaires pour trouver $\boldsymbol{s}_t$ à chaque tour t, étant donné la séquence de feedbacks observés jusqu'à présent.

1.1 Travaux Relatifs

Dans la littérature des jeux de prédiction combinatoire, trois principales stratégies ont été proposées pour atteindre un regret espéré qui est sous-linéaire en l'horizon de jeu T, et polynomial en la dimension d de l'espace de décision. La première, et sans doute la plus simple des stratégies, s'appelle *Follow the Perturbed Leader* (FPL) : à chaque tour t, l'apprenant choisit aléatoirement un vecteur de perturbation $\boldsymbol{z}_t \in \mathbb{R}^d$, et sélectionne un minimiseur de $\eta \boldsymbol{L}_t + \boldsymbol{z}_t$, où $\eta \in (0, 1]$ est un paramètre d'apprentissage et $\boldsymbol{L}_t$ est la perte cumulative : $\boldsymbol{L}_t = \boldsymbol{\ell}_1 + \cdots + \boldsymbol{\ell}_{t-1}$. Introduit par Hannan [15], et amélioré dans [18, 19], l'algorithme FPL atteint un regret espéré de $\mathcal{O}(d^{\frac{3}{2}}\sqrt{T})$.

La seconde stratégie est fondée sur le prédicteur "exponentially weighted average" bien connu dans le cadre de la prédiction avec avis d'experts [6]. L'idée générale est de maintenir un poids pour chaque solution faisable $\boldsymbol{s} \in \mathcal{S}$, qui décroit exponentiellement avec la perte cumulative de $\boldsymbol{s}$. Spécifiquement, à chaque tour t, l'apprenant choisit une solution $\boldsymbol{s}_t \in \mathcal{S}$ aléaoirement selon la distribution exponentielle $p_t(\boldsymbol{s}) \sim \exp(-\eta \langle \boldsymbol{L}_t, \boldsymbol{s} \rangle)$. Cette stratégie, appelée *Expanded Hedge* (EH) dans [20], atteint un regret espéré de $\mathcal{O}(d^{\frac{3}{2}}\sqrt{T})$, similaire à celui de FPL.

Enfin, la troisième stratégie appelée *Follow the Regularized Leader*, est le paradigme principalement utilisé en optimisation convexe en-ligne [16]. L'apprenant utilise ici l'enveloppe convexe de $\mathcal{S}$, notée $\mathrm{conv}(\mathcal{S})$. A chaque tour t, l'apprenant démarre en choisissant un point $\boldsymbol{p}_t \in \mathrm{conv}(\mathcal{S})$ qui minimize $\eta \langle \hat{\boldsymbol{L}}_t, \boldsymbol{p} \rangle + F(\boldsymbol{p})$, où F est une fonction de régularisation. Ensuite, $\boldsymbol{p}_t$ est décomposé en une combinaison convexe de solutions faisables dans $\mathcal{S}$, à partir de laquelle une décision $\boldsymbol{s}_t$ est choisie aléatoirement. Pour les fonctions de perte linéaire, cette stratégie est équivalent à l'algorithme *Online Stochastic Mirror Descent* (OSMD) [2, 30] qui, à chaque tour de jeu, accomplit une descente de gradient dans l'espace dual de $\mathrm{conv}(\mathcal{S})$ selon F, et projette ensuite cette solution sur l'espace primal, selon la divergence de Bregman définie sur F. En particulier, lorsque F est le régularisateur euclidien, OSMD coïncide avec la descente de gradient stochastique (SGD) [31]. Lorsque F est le régularisateur entropique, OSMD correspond à l'algorithme *Component Hedge* (CH) [20], qui atteint un regret espéré *optimal* de $\mathcal{O}(d\sqrt{T})$.

Si l'on se focalise sur la mesure de regret, l'état de l'art nous indique que peu d'innovations restent à faire dans les jeux à information totale. Cependant, le constat est bien différent dès lors que l'on tient compte des aspects calculatoires : les trois stratégies mentionnées plus haut font intervenir de puissants oracles pour prendre des décisions dans des espaces $\mathcal{S}$ représentés par des contraintes combinatoires. En particulier, la stratégie EH nécessite, à chaque itération, d'échantillonner une solution selon une famille exponentielle sur $\mathcal{S}$, ce qui est un problème #P-difficile [12]. De manière analogue, la stratégie FPL doit répétitivement résoudre un problème d'optimisation linéaire sur $\mathcal{S}$, ce qui est généralement NP-difficile [8]. Pour l'algorithme OSMD et ses spécialisations SGD et CH, la difficulté calculatoire est exacerbée par le fait que, même si l'apprenant a accès à un oracle d'optimisation linéaire, il doit accomplir à chaque tour de jeu une projection de Bregman pour laquelle les meilleurs algorithmes ont une complexité temporelle en $\mathcal{O}(d^6)$ [33] ou $\mathcal{O}(d^4)$ [24].

Bien que les jeux de prédiction combinatoire soient généralement NP-difficiles, il est dans certains cas possible d'obtenir des implémentations efficaces pour les oracles d'échantillonnage et d'optimisation. Par exemple, lorsque l'espace de décision $\mathcal{S}$ coïncide avec les bases d'un matroïde binaire, ou les appariements parfaits d'un graphe biparti, l'optimisation linéaire peut être résolue en temps polynomial, et donc des implémentations efficaces de FPL et OSMD peuvent être dérivées [17, 20, 34, 30]. D'autre part, lorsque les solutions faisables de $\mathcal{S}$ correspondent aux chemins d'un graphe orienté sans circuit (DAG), l'oracle d'échantillonnage peut être simulé par la technique de *weight pushing* [25], qui évalue récursivement la constante de partition d'une famille exponentielle sur les arêtes du DAG. A partir de cette technique, des implémentations efficaces de EH peuvent être dérivées [35, 29].

1.2 Nos Résultats

Envisager les solutions faisables comme des chemins d'un DAG n'est qu'une des nombreuses abstractions possibles qui ont été proposées dans la littérature en complexité de circuits, pour représenter des espaces combinatoires. En compilation de connaissances [11], diverses classes de circuits booléens ont été identifiées, chacune associée avec une collection de tâches d'inférence qui peuvent être accomplies en temps polynomial. Ces résultats théoriques nous incitent naturellement à nous poser la question suivante : est-il possible de *compiler* un ensemble de contraintes représentant un espace combinatoire $\mathcal{S}$ en un circuit booléen compact, pour lequel l'échantillonnage de solution et l'optimisation

linéaire sont traitables ? En regardant le processus de compilation comme une étape de "pré-processing", nous pouvons obtenir des implémentations efficaces des oracles d'échantillonnage et d'optimisation, dès lors que la taille du circuit compilé n'est pas trop grande.

Cet article vise à résoudre des jeux de prédiction combinatoire en compilant les espaces de décision en circuit dDNNF (*deterministic Decomposable Negation Normal Form*) [9]. Pour cette classe de circuits, il existe des compilateurs génériques prenant en entrée une formule SAT représentant un espace de décision $\mathcal{S}$, et retournant en sortie un circuit C qui code $\mathcal{S}$ [10, 23]. Bien que la taille de C puisse croître de manière exponentielle en la "treewidth" de la formule d'entrée, elle est souvent bien plus petite en pratique ; les compilateurs actuels sont capables de compresser des espaces combinatoires définis sur des milliers de variables et de contraintes.

Avec ces outils de compilation en main, nos contributions sont les suivantes : (i) nous montrons que pour les circuits dDNNF, l'oracle d'échantillonnage pour EH, et l'oracle d'optimisation linéaire pour FPL, prennent tous deux un temps linéaire en utilisant de simples variantes de la technique weight-pushing ; (ii) pour les stratégies SGD and CH, nous développons une méthode de projection-décomposition de Bregman qui utilise seulement $\mathcal{O}(d^2 \ln(dT))$ appels à l'oracle d'optimisation linéaire ; (iii) nous montrons expérimentalement sur des tâches de configuration séquentielle et de planification en-ligne que EH et FPL sont rapides, mais nos variants de SGD et CH sont plus efficaces pour minimiser le regret empirique.

2 Inférence Efficace

Pour les jeux de prédiction combinatoire considérés dans ce papier, nous supposons que l'espace de décision $\mathcal{S}$ est défini à partir d'un ensemble fini d'attributs binaires, et nous utilisons $X = \{x_1, \cdots, x_d\}$ pour caractériser l'ensemble de toutes les paires "attribut-valeur", appelées littéraux ou *indicateurs*. Une *solution* est un vecteur $\boldsymbol{s} \in \{0,1\}^d$ tel que $s(i) + s(j) = 1$ pour chque paire d'indicateurs distincts $x_i, x_j \in X$ définis sur le même attribut. Ainsi, $\|\boldsymbol{s}\|_1 = \frac{d}{2}$ pour toute solution $\boldsymbol{s} \in \mathcal{S}$.

Un circuit NNF sur X est un DAG enraciné, dont les nœuds internes sont étiquetés par $\vee$ (or-node) ou $\wedge$ (and-node), et dont les feuilles sont étiquetées par un indicateur dans X, ou bien une constante dans $\{0,1\}$. La taille de C, notée $|C|$, est donnée par le nombre d'arêtes dans C. L'ensemble des attributs apparaissant dans le sous-graphe de C, enraciné au nœud c, est noté $att(c)$.

Pour des raisons de clarté, nous supposerons que tout circuit NNF C satisfait deux propriétés : (i) tout nœud interne c dans C a exactement deux enfants, notés c_l et c_r, et (ii) $att(c_l) = att(c_r) \neq \varnothing$ pour tout or-node c de C. Un circuit NNF satisfaisant ces deux conditions est dit *lisse*. Comme il est montré dans [9], tout circuit booléen C peut être transformé en un circuit NNF lisse équivalent, dont la taille est linéaire en $|C|$.

Q	R	$\oplus$	$\otimes$	$\top$	$\bot$
maxmin	$\{0,1\}$	max	min	1	0
minsum	$\mathbb{R} \cup \{+\infty\}$	min	$+$	0	$+\infty$
sumprod	$\mathbb{R}$	$+$	$*$	1	0

TABLE 1 – Semi-anneaux commutatifs

En voyant les indicateurs comme des "portes d'entrées" et les nœuds comme des "portes de sortie", nous pouvons définir diverses tâches d'inférence sur les circuits booléens, qui dépendent du type d'entrée et de la sémantique des nœuds. Comme le suggèrent Friesen & Domingos pour les fonctions sommes-produits [13], les tâches d'inférence peuvent être caractérisées par des opérations de semi-anneaux. Rappelons ici qu'un *semi-anneaux commutatif* est un tuple $(R, \oplus, \otimes, \bot, \top)$ tel que R est un ensemble contenant les éléments $\bot$ et $\top$, $\oplus$ est un opérateur binaire associatif et commutatif sur R avec l'élément neutre $\bot$, $\otimes$ est un opérateur binaire associatif sur R avec l'élément neutre $\top$ et l'élément absorbant $\bot$, et l'opérateur $\otimes$ se distribue (à droite et à gauche) sur l'opérateur $\oplus$.

Les tâches d'inférence sur un circuit NNF C sont spécifiées par le choix d'un semi-anneau commutatif $Q = (R, \oplus, \otimes, \bot, \top)$ et d'un vecteur d'entrée $\boldsymbol{w} \in R^d$. La *sortie* d'un nœud c dans C pour Q étant donné $\boldsymbol{w}$ est notée $Q(c \,|\, \boldsymbol{w})$, et elle est récursivement donnée par

$$Q(c \,|\, w) = \begin{cases} w(i) & \text{si } c \text{ est un indicateur } x_i, \\ \top & \text{si } c \text{ est la constante } 1, \\ \bot & \text{si } c \text{ est la constante } 0, \\ Q(c_l \,|\, \boldsymbol{w}) \oplus Q(c_r \,|\, \boldsymbol{w}) & \text{si } c \text{ est le nœud } \vee, \\ Q(c_l \,|\, \boldsymbol{w}) \otimes Q(c_r \,|\, \boldsymbol{w}) & \text{si } c \text{ est le nœud } \wedge \end{cases}$$

Nous notons $Q(C \,|\, \boldsymbol{w})$ la sortie de la racine de C pour Q selon $\boldsymbol{w}$. Les semi-anneaux commutatifs jouant un rôle majeur dans cette étude sont décrits dans le tableau 1 ; maxmin, minsum, and sumprod, sont utilisés pour capturer les tâches d'inférence reliées respectivement à l'évaluation de modèle, l'optimisation linéaire, et l'échantillonnage de modèle.

2.1 Évaluation de Modèle

Étant donné un circuit NNF C sur X, l'évaluation de modèle (*model checking*) est de décider si une solution possible $\boldsymbol{s} \in \{0,1\}^d$ est vraie dans C selon la sémantique propositionnelle des nœuds. Il est clair que $\boldsymbol{s}$ est un modèle de C ssi $\mathrm{maxmin}(C \mid \boldsymbol{s}) = 1$, ce qui peut être déterminé en temps $\mathcal{O}(|C|)$. Un circuit NNF C est appelé *représentation* d'un ensemble de solutions faisables $\mathcal{S} \subseteq \{0,1\}^d$ si $sol(C) = \mathcal{S}$, où $sol(C)$ est l'ensemble des modèles de C.

Si l'on excepte l'évaluation de modèle, la totalité des tâches d'inférence dans les circuits NNF sont NP-difficiles. En effet, le langage NNF couvre la classe des formules SAT. Ainsi, nous avons besoin de contraindre cette

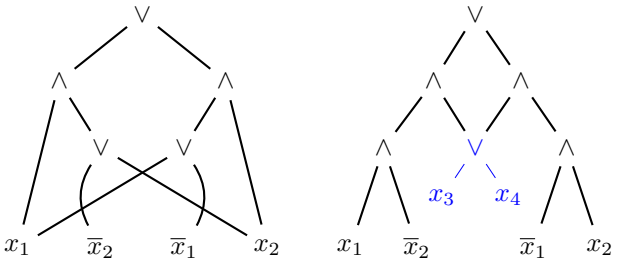


FIGURE 1 – A gauche, un circuit DNNF et à droite un circuit dDNNF.

Algorithm	Regret	Runtime
EH	$\mathcal{O}(d^{\frac{3}{2}}\sqrt{T})$	$\mathcal{O}(\lvert C\rvert)$
FPL	$\mathcal{O}(d^{\frac{3}{2}}\sqrt{T})$	$\mathcal{O}(\lvert C\rvert)$
SGD+PCG	$\mathcal{O}(d(\sqrt{T}+\ln T))$	$\mathcal{O}(d^2\lvert C\rvert \ln T)$
δ-CH+PCG	$\mathcal{O}(d(\sqrt{T}+\ln T))$	$\mathcal{O}\left(\frac{d^2\lvert C\rvert}{\delta}\ln\frac{T}{\delta}\right)$

TABLE 2 – Regret espéré et complexité temporelle (par tour) de chaque stratégie implémentée sur un circuit dDNNF C.

classe pour obtenir des formes efficaces d'optimisation et d'échantillonnage.

2.2 Décomposabilité et Optimisation

Un circuit booléen C est *décomposable* si pour tout and-node c de C, nous avons $att(c_l) \cap att(c_r) = \varnothing$. La classe des circuits NNF décomposables est notée DNNF. Pour de tels circuits, qui sont similaires aux réseaux sommes-produits [28], nous pouvons obtenir une implémentation efficace de l'oracle d'optimisation linéaire.

Proposition 1. Soit $\mathcal{S} \subseteq \{0,1\}^d$ un espace de décision non vide représenté par un circuit DNNF C, et soit $\boldsymbol{w} \in \mathbb{R}^d$ une fonction objectif linéaire. Trouver un minimiseur de $\boldsymbol{w}$ dans $\mathcal{S}$ peut être réalisé en temps $\mathcal{O}(|C|)$.

2.3 Déterminisme et Échantillonnage

Comme le problème de comptage du nombre de modèles d'un circuit DNNF est #P-difficile [11], nous avons besoin de spécialiser cette classe afin d'obtenir une implémentation efficace de l'oracle d'échantillonnage. Dans cette perspective, un circuit NNF C est dit *déterministe* si $\mathrm{minmax}(c_l \mid \boldsymbol{s}) + \mathrm{minmax}(c_r \mid \boldsymbol{s}) \leq 1$ pour tout or-node $c \in C$ et toute solution faisable $\boldsymbol{s}$. La classe des circuits DNNF déterministes est notée dDNNF.

Proposition 2. Soit $\mathcal{S} \subseteq \{0,1\}^d$ un espace de décision représenté par un circuit dDNNF circuit C. Pour un vecteur $\boldsymbol{w} \in \mathbb{R}^d$, soit $\mathbb{P}_{\boldsymbol{w}}$ la famille exponentielle sur $\mathcal{S}$ donnée par :

$$\mathbb{P}_{\boldsymbol{w}}(\boldsymbol{s}) = \frac{\exp\langle \boldsymbol{w}, \boldsymbol{s}\rangle}{\sum_{\boldsymbol{s}'\in\mathcal{S}} \exp\langle \boldsymbol{w}, \boldsymbol{s}'\rangle}$$

L'échantillonnage $\boldsymbol{s} \sim \mathbb{P}_{\boldsymbol{w}}$ peut être fait en temps $\mathcal{O}(|C|)$.

3 Prédiction Efficace

Après une excursion dans les langages de compilation, nous sommes maintenant prêts à examiner des caractérisations efficaces de stratégies de prédiction combinatoire. Nos résultats sont résumés dans le tableau 2. En utilisant le fait que $\|\boldsymbol{s}\|_1 = d/2$, les bornes de regret pour EH and FPL peuvent facilement être dérivées de [1] and [18]. Les deux stratégies sont aussi faciles à implémenter sur des dDNNF. En effet, rappelons que EH tire, à chaque tour t, une solution faisable $\boldsymbol{s}_t \in \mathcal{S}$ aléatoirement selon la distribution $\mathbb{P}_{-\eta \boldsymbol{L}_t}$, où $\boldsymbol{L}_t = \boldsymbol{\ell}_1 + \cdots + \boldsymbol{\ell}_{t-1}$. Ainsi, par application directe de la proposition 2, cette stratégie s'exécute en temps $\mathcal{O}(|C|)$ par tour, dès lors que l'espace $\mathcal{S}$ a été compilé au préalable en un circuit dDNNF C. Pour la stratégie FPL, chaque tour t est accompli en choisissant un minimiseur $\boldsymbol{s}_t \in \mathcal{S}$ de la fonction objectif $\eta \boldsymbol{L}_t - \boldsymbol{z}_t$, où $\boldsymbol{z}_t \in \mathbb{R}^d$ est un vecteur de perturbation dont les composants sont des variables aléatoires exponentielles indépendantes. Par application de la proposition 1, FPL s'exécute en temps $\mathcal{O}(|C|)$ par tour, en utilisant un codage dDNNF C de $\mathcal{S}$, et le fait que $|C|$ est en $\Omega(d)$.

Cependant, la stratégie OSMD, et ses spécialisations SGD et CH, requièrent plus d'attention du fait qu'elles utilisent une coûteuse étape de projection-décomposition à chaque tour de jeu.

3.1 Descente Miroir Stochastique

L'idée générale de l'algorithme Online Mirror Descent (OMD) est de "suivre le leader régularisé" par le biais d'une approche primale-duale [26, 4]. Soit $\mathcal{K}$ un ensemble convexe, et notons $int(\mathcal{K})$ son intérieur. Étant donné une fonction de régularisation F définie sur $\mathcal{K}$, OMD réalise de manière itérative une descente de gradient dans l'intérieur de l'espace dual $\mathcal{K}^*$, et projette le point dual sur l'espace primal $\mathcal{K}$. La connexion entre $\mathcal{K}$ et $\mathcal{K}^*$ est assurée par les gradients ∇F et ∇F^*, où F^* est le conjugué convexe de F défini sur $\mathcal{K}^*$. L'étape de projection est capturée par la divergence de Bregman de F, qui est une fonction de la forme $B_F \colon \mathcal{K} \times \mathrm{int}(\mathcal{K}) \to \mathbb{R}$ donnée par :

$$B_F(\boldsymbol{p}, \boldsymbol{q}) = F(\boldsymbol{p}) - F(\boldsymbol{q}) - \langle \nabla F(\boldsymbol{q}), \boldsymbol{p} - \boldsymbol{q}\rangle$$

Dans le variante stochastique de OMD, introduite par Audibert et. al. [1, 2], et spécifiée dans l'algorithme 1, chaque étape de projection est réalisée sur le sous-ensemble $\mathrm{conv}(\mathcal{S})$ of $\mathcal{K}$, et le point de projection $\boldsymbol{p}_t$ est décomposé en une combinaison convexe de solutions faisables dans $\mathcal{S}$, à partir de laquelle une solution est tirée aléatoirement pour l'étape de prédiction.

Pour les régularisateurs habituels, le gradient $\nabla F(\boldsymbol{p}_t)$ et son dual $\nabla F^*(\boldsymbol{u}_t)$ sont facilement calculables, et nous supposerons ici que le temps dépensé pour leur construction est négligeable par rapport au temps d'exécution de l'oracle

Algorithme 1 OSMD

Entrée : espace $\mathcal{S} \subseteq \{0,1\}^d$, horizon $T \in \mathbb{Z}_+$
Paramètres : régularisateur F sur $\mathcal{K} \supseteq \text{conv}(\mathcal{S})$, scalaire $\eta \in (0,1]$

soit $\boldsymbol{u}_1 = \mathbf{0}$
pour $t = 1$ **to** T **faire**
  soit $\boldsymbol{p}_t \in \text{Argmin}_{\boldsymbol{p} \in \text{conv}(\mathcal{S})} B_F(\boldsymbol{p}, \nabla F^*(\boldsymbol{u}_t))$
  jouer $\boldsymbol{s}_t \sim \boldsymbol{p}_t$ et observer $\boldsymbol{\ell}_t$
  choisir $\boldsymbol{u}_{t+1} = \nabla F(\boldsymbol{p}_t) - \eta \boldsymbol{\ell}_t$
end pour

Algorithme 2 PCG

Entrée : $\mathcal{S} \subseteq \{0,1\}^d$, $f\colon \mathcal{K} \to \mathbb{R}$, $n \in \mathbb{Z}_+$
Paramètres : scalaires $\{\eta_j\}_{j=1}^n$

soit $\boldsymbol{p}_1$ un point arbitraire de $\mathcal{S}$
pour $j = 1$ **to** n **faire**
  soit $\sum_{i=1}^j \alpha_i \boldsymbol{s}_i$ une décomposition convexe de $\boldsymbol{p}_j$
  soit $\boldsymbol{s}_j^+ \in \text{Argmin}_{\boldsymbol{p} \in \text{conv}(\mathcal{S})} \langle \nabla f(\boldsymbol{p}_j), \boldsymbol{p} \rangle$
  soit $\boldsymbol{s}_j^- \in \text{Argmin}_{\boldsymbol{s} \in \{\boldsymbol{s}_1, \cdots, \boldsymbol{s}_j\}} \langle -\nabla f(\boldsymbol{p}_j), \boldsymbol{s} \rangle$
  choisir $\boldsymbol{p}_{j+1} = \boldsymbol{p}_j + \eta_j(\boldsymbol{s}_j^+ - \boldsymbol{s}_j^-)$
end pour

d'optimisation. En fait, le goulot d'étranglement de OSMD est de trouver un minimiseur $\boldsymbol{p}_t$ de $B_F(\boldsymbol{p}, \nabla F^*(\boldsymbol{u}_t))$ dans l'enveloppe convexe de $\mathcal{S}$, et de décomposer $\boldsymbol{p}_t$ en une combinaison convexe de solutions dans $\mathcal{S}$. Heureusement, selon des hypothèses raisonnables sur la courbure de B_F, cette étape de projection-décomposition peut être calculée efficacement, en s'inspirant de résultats récents sur les algorithmes d'optimisation convexe "projection-free".

De manière plus formelle, nous disons qu'une divergence de Bregman B_F possède le *nombre conditionnel* β/α si, en son premier argument, B_F est α-fortement convexe et β-lisse [1] selon la norme euclidienne $\|\cdot\|_2$. Pour de tels régularisateurs, le résultat suivant établit que l'étape de projection-décomposition peut être approximée efficacement, en exploitant la méthode *Pairwise Conditional Gradient* (PCG), qui est une variante de l'algorithme Frank-Wolfe dont la vitesse de convergence a été analysée dans [22, 14, 3].

Lemme 1. Soit $\mathcal{S} \subseteq \{0,1\}^d$ un ensemble de décision représenté par un circuit dDNNF C, et F un régularisateur sur $\mathcal{K} \supseteq \text{conv}(\mathcal{S})$ tel que B_F possède le nombre conditionnel β/α. Alors, pour tout $\boldsymbol{q} \in \text{int}(\mathcal{K})$ et $\epsilon \in (0,1)$, on peut trouver en temps $\mathcal{O}(\frac{\beta}{\alpha} \ln \frac{\beta d}{\epsilon} d^2 |C|)$ une décomposition convexe de $\boldsymbol{p} \in \text{conv}(\mathcal{S})$ telle que

$$B_F(\boldsymbol{p}, \boldsymbol{q}) - \min_{\boldsymbol{p}' \in \text{conv}(\mathcal{C})} B_F(\boldsymbol{p}', \boldsymbol{q}) \leq \epsilon$$

Notons OSMD+PCG la version raffinée de l'algorithme OSMD qui utilise PCG à chaque tour t, afin d'approximer l'étape de projection-d"composition. En plus du régularisateur F et du paramètre η, OSMD+PCG prend en entrée une séquence $\{\epsilon_t\}_{t=1}^T$ telle que

$$B_F(\boldsymbol{p}_t, \boldsymbol{q}_t) - B_F(\boldsymbol{p}_t^*, \boldsymbol{q}_t) \leq \epsilon_t$$

où $\boldsymbol{p}_t$ est le point retourné par PCG, $\boldsymbol{q}_t = \nabla F_\phi^*(\boldsymbol{u}_t)$, et $\boldsymbol{p}_t^*$ est le minimiseur de $B_F(\boldsymbol{p}, \boldsymbol{q}_t)$ sur $\text{conv}(\mathcal{S})$.

Théorème 1. Supposons que OSMD+PCG prenne en entrée un encodage dDNNF C d'un espace de décision $\mathcal{S} \subseteq \{0,1\}^d$, ainsi qu'un horizon T, et utilise un régularisateur F sur $\mathcal{K} \supseteq \text{conv}(\mathcal{S})$ tel que B_F possède le nombre conditionnel β/α, avec le paramètre d'apprentissage $\eta \in (0,1]$ et une séquence de valeurs de précision $\{\epsilon_t\}_{t=1}^T$ telle que $\epsilon_t = \gamma/t^2$ pour $\gamma > 0$. Alors OSMD+PCG atteint le regret espéré :

$$\begin{aligned} R_T \leq \sqrt{\frac{2\gamma d}{\alpha}}(\ln T + 1) + \frac{1}{\eta} \max_{\boldsymbol{s} \in \mathcal{S}} B_F(\boldsymbol{s}, \boldsymbol{p}_1^*) \\ + \frac{1}{\eta} \sum_{t=1}^T B_{F^*}(\nabla F(\boldsymbol{p}_t^*) - \eta \boldsymbol{\ell}_t, \nabla F(\boldsymbol{p}_t)) \end{aligned} \tag{1}$$

avec une complexité temporelle en $\mathcal{O}\left(\frac{\beta}{\alpha} \ln \frac{\beta d T}{\gamma} d^2 |C|\right)$.

3.2 Descente de Gradient Stochastique

L'algorithme (en-ligne) SGD est dérivé de OSMD en utilisant le régularisateur euclidien $F(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{p}\|_2^2$. Dans ce cadre simple, l'espace primal et l'espace dual coïncident avec $\mathbb{R}^d$, et donc, $F^*(\boldsymbol{u}) = \boldsymbol{u}$, $\nabla F(\boldsymbol{p}) = \boldsymbol{p}$, et $\nabla F^*(\boldsymbol{u}) = \boldsymbol{u}$. De plus, B_F a le nombre conditionnel $1/1$, puisque $B_F(\boldsymbol{p}, \boldsymbol{q}) = \frac{1}{2}\|\boldsymbol{p} - \boldsymbol{q}\|_2^2$. Nous notons ici SGD+PCG l'instance de OSMD+PCG définie sur le régularisateur euclidien.

Proposition 3. L'algorithme SGD+PCG atteint un regret espéré de $d(\sqrt{T} + \ln T + 1)$ avec une complexité temporelle par tour de $\mathcal{O}(d^2|C| \ln T)$ en utilisant $\eta = 1/\sqrt{T}$ et $\gamma = d/2$.

3.3 Component Hedge

L'algorithme CH est dérivé de OSMD en utilisant le régularisateur entropique $F(\boldsymbol{p}) = \sum_{i=1}^d p(i)(\ln p(i) - 1)$, pour lequel le conjugué est $F^*(\boldsymbol{u}) = \sum_{i=1}^d \exp u(i)$. Ici, on ne peut pas trouver un nombre conditionnel fini pour la divergence associée $B_F(\boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^d p(i) \ln \frac{p(i)}{q(i)} - (p(i) - q(i))$, puisque son gradient n'est pas borné. Cette difficulté peut cependant être contournée en utilisant une astuce suggérée dans [21]. L'idée consiste à remplacer le régularisateur entropique par la fonction $F_\delta(\boldsymbol{p}) = F(\boldsymbol{p} + \boldsymbol{\delta})$, où $\delta \in (0,1)$ et $\boldsymbol{\delta} = (\delta, \cdots, \delta)$. Pour cette fonction l'espace primal est $(-\delta, +\infty)$, et puisque $F_\delta^*(\boldsymbol{u}) = F^*(\boldsymbol{u}) - \langle \boldsymbol{u}, \boldsymbol{\delta} \rangle$, l'espace dual est $\mathbb{R}^d$. Il est facile de montrer que

$$\frac{\partial F_\delta(\boldsymbol{p})}{\partial p(i)} = \ln(p(i) + \delta) \qquad \frac{\partial F_\delta^*(\boldsymbol{u})}{\partial u(i)} = e^{u(i)} - \delta$$
$$B_{F_\delta}(\boldsymbol{p}, \boldsymbol{q}) = B_F(\boldsymbol{p} + \boldsymbol{\delta}, \boldsymbol{q} + \boldsymbol{\delta}) \quad B_{F_\delta}^*(\boldsymbol{u}, \boldsymbol{v}) = B_F^*(\boldsymbol{u}, \boldsymbol{v})$$

1. Ces propriétés de courbure sont habituelles en optimisation convexe [16].

où $B_F^*(\boldsymbol{u}, \boldsymbol{v}) = \sum_{i=1}^{d} e^{v(i)}(e^{v(i)-u(i)} + v(i) - u(i) - 1)$. Enfin, puisque les dérivées partielles premières et secondes de $B_{F_\delta}^*(\boldsymbol{p}, \boldsymbol{q})$ à la coordonnée $p(i)$ sont

$$\frac{\partial B_{F_\delta}(\boldsymbol{p}, \boldsymbol{q})}{\partial p(i)} = \ln \frac{p(i) + \delta}{q(i) + \delta} \quad \frac{\partial^2 B_{F_\delta}(\boldsymbol{p}, \boldsymbol{q})}{\partial^2 p(i)} = \ln \frac{1}{p(i) + \delta}$$

il suit que B_{F_δ} possède le nombre conditionnel $1+\delta/\delta$. En effet, étant donné un point arbitraire $\boldsymbol{q} \in \mathrm{int}(-\delta, +\infty)$, soit $H_{\boldsymbol{q}}(\boldsymbol{p})$ la matrice hessienne de $B_{F_\delta}(\boldsymbol{p}, \boldsymbol{q})$ at $\boldsymbol{p} \in \mathrm{conv}(\mathcal{S})$. Alors, pour tout $\boldsymbol{z} \in \mathbb{R}^d$, les entrées de la diagonale de $H_{\boldsymbol{q}}(\boldsymbol{p})$ satisfont

$$\frac{1}{1+\delta} \leq \frac{\partial^2 B_F(\boldsymbol{p}, \boldsymbol{q})}{\partial^2 p(i)} z(i)^2 \leq \frac{1}{\delta}$$

en utilisant le fait que $p(i) \in [0, 1]$. Ainsi, $\alpha \boldsymbol{I} \preccurlyeq H_{\boldsymbol{q}}(\boldsymbol{p}) \preccurlyeq \beta \boldsymbol{I}$ pour $\alpha = 1/1+\delta$ et $\beta = 1/\delta$. Dans la suite, l'instance de OSMD+PCG qui utilise F_δ comme régularisateur est notée δ-CH+PCG.

Proposition 4. L'algorithme δ-CH+PCG atteint un regret espéré de $d(1 + 2\delta)(\sqrt{T} + \ln T + 1)$ avec une complexité temporelle par tour de $\mathcal{O}\left(d^2|C|/\delta \ln T/\delta\right)$, en utilisant $\eta = 1/\sqrt{T}$ et $\gamma = 2d(1/2 + \delta)/(1 + \delta)$.

4 Expérimentations

Afin d'évaluer la performance des différentes stratégies examinées dans la section précédente, nous avons considéré 15 instances de la SAT Library [2], décrites dans le tableau 3. Les cinq premières rangées sont des problèmes de configuration (automobile) et les rangées suivantes sont des tâches de planification. Les quatre premières colonnes du tableau indiquent le nom de l'instance SAT, le nombre d'attributs ($d/2$), le nombre de contraintes ($|\mathtt{SAT}|$), et le nombre de solutions faisables $|\mathcal{S}|$. Nous avons utilisé le compilateur D4 [3] [23] pour transformer les instances SAT en circuits dDNNF. La taille $|C|$ du circuit compilé est reportée dans la cinquième colonne.

Afin de simuler des jeux de prédiction combinatoire, nous avons utilisé le protocole suivant. Supposons que l'ensemble $X = \{x_1, \cdots, x_d\}$ des indicateurs soit trié par ordre lexicographique, de telle manière que pour tout entier impair i, le couple (x_i, x_{i+1}) code les deux configurations du même attribut. Nous construisons d'abord un vecteur $\boldsymbol{\mu}_0$ de $d/2$ variables de Bernoulli indépendantes. Pour chaque tour t du jeu, $\boldsymbol{\mu}_t$ est fixé à $\boldsymbol{\mu}_{t-1}$ avec probabilité 0.9, où tiré aléatoirement et uniformément sur $[0, 1]^{d/2}$ avec probabilité 0.1. Le feedback fourni à l'apprenant est un vecteur de perte $\boldsymbol{\ell}_t \in \{0, 1/d\}^d$ tel que $\ell_t(i) + \ell_t(i+1) = 1/d$, et $\ell_t(i) = 1/d$ avec probabilité $\mu_t(i+1/2)$ pour chaque entier impair i. Ainsi, $\ell_t(i+1) = 1/d$ avec probabilité $1 - \mu_t(i+1/2)$. Bien que ce protocole soit essentiellement stochastique, l'environnement change (en secret) $\boldsymbol{\mu}_t$ avec probabilité 0.1 à chaque tour, afin de déjouer l'apprenant.

2. www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html
3. www.cril.univ-artois.fr/KC/d4.html

Avant d'examiner les résultats, donnons quelques indications sur le choix des paramètres. Pour les algorithmes FPL et EH, nous avons utilisé les valeurs de η reportées dans [1] et [18]. Concernant SGD+PCG et δ-CH+PCG, nous avons utilisé les valeurs de η et γ déterminées par notre analyse théorique : les valeurs de $\{\eta_t\}$ pour PCG ont été calculées par recherche binaire, comme il est préconisé dans [14], et la valeur de δ a été fixée à $1/\ln d$ afin de garder une complexité temporelle quadratique pour δ-CH+PCG. Enfin, nous avons fixé l'horizon T à 10^3, et nous avons fixé une journée pour le temps limite total de calcul pour l'apprentissage.

Dans nos expérimentations, le regret est mesuré par la différence de perte cumulative entre la stratégie étudiée et la meilleure solution faisable, obtenue par recul sur le jeu à l'horizon T. Cette mesure établie en moyenne sur 10 simulations, et divisée par T pour donner un regret empirique moyen. Les résultats correspondants sont donnés dans les quatre dernières colonnes du tableau 3. Le symbole "$-$" indique que l'apprenant n'a pas réussi à accomplir les T tours en une journée. Concernant le regret, SGD+PCG et δ-CH+PCG sont meilleurs que EH et FPL, ce qui confirme nos résultats théoriques. Nous mentionnons au passage que SGD+PCG et δ-CH+PCG sont remarquablement stables. Par contraste, FPL exhibe une grande variance.

Concernant les temps de calcul, EH et FPL sont sans surprise plus rapides que SGD+PCG et δ-CH+PCG. Pour les très grandes instances c129-fr, c140-fc, c163-fw, et log-5, nous avons constaté que EH et FPL réalisent chaque tour sur approximativement une minute, alors que les algorithmes OSMD+PCG requièrent plusieurs minutes par tour, à cause des ressources dépensées pour approximer la projection de Bregman. Cependant, il est important de souligner que la vitesse de convergence de PCG est, en pratique, bien plus rapide que l'analyse théorique en $\tilde{\mathcal{O}}(d^2|C|)$. En fait, SGD+PCG et δ-CH+PCG peuvent traiter pratiquement toutes les instances de problèmes de planification en moins d'une minute par tour. Nous avons observé que, pour la plupart des instances, SGD+PCG est légèrement plus rapide que δ-CH+PCG ; c'est particulièrement le cas pour les grands domaines où les faibles valeurs de δ ont un impact sur le temps de calcul. En résumé, SGD+PCG offre le meilleur compromis entre précision et temps de calcul ; puisque toutes les solutions sont denses ($\|\boldsymbol{s}\|_1 = d/2$), la différence de précision entre SGD+PCG et δ-CH+PCG n'est pas significative.

5 Perspectives

En lumière des résultats obtenus dans cette étude, une perspective naturelle de recherche est d'étendre notre cadre à d'autres classes de jeux de prédiction combinatoire. Notamment, le mode *semi-bandit* semble à portée de main. L'extension de EH en mode-semi-bandit, appelée EXP2 [2], utilise un ajustement des poids pour estimer le vecteur de perte à chaque itération. Par simple application de la proposition 2, cet ajustement peut être calculé en temps linéaire sur des circuits dDNNF. De manière analogue, l'extension semi-bandit de FPL exploite une méthode

Name	$d/2$	$\lvert$SAT$\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert C\rvert$	EH	FPL	SGD+PCG	δ-CH+PCG
c129-fr	1888	6245	$1.33\ 10^{166}$	$1.42\ 10^{6}$	1023 ± 181 (65s)	1435 ± 321 (62s)	-	-
c140-fc	1828	4267	$5.74\ 10^{151}$	$6.94\ 10^{5}$	864 ± 175 (22s)	915 ± 185 (22s)	-	-
c163-fw	1815	3580	$2.97\ 10^{140}$	$8.93\ 10^{5}$	798 ± 87 (24s)	972 ± 104 (21s)	-	-
c169-fv	1411	637	$3.22\ 10^{15}$	$7.20\ 10^{2}$	18 ± 6 (<1s)	26 ± 10 (<1s)	9 ± 1 (4s)	9 ± 1 (5s)
4-step	165	396	$8.34\ 10^{4}$	$1.29\ 10^{2}$	3 ± 2 (<1s)	5 ± 3 (<1s)	2 ± 1 (2s)	2 ± 1 (2s)
5-step	177	459	$8.13\ 10^{4}$	$5.80\ 10^{1}$	3 ± 1 (<1s)	3 ± 1 (<1s)	1 ± 0.9 (<1s)	1 ± 0.8 (<1s)
log-1	939	3742	$5.64\ 10^{20}$	$9.43\ 10^{2}$	46 ± 5 (<1s)	51 ± 8 (<1s)	7 ± 3 (4s)	7 ± 1 (5s)
log-2	1337	24 735	$3.23\ 10^{10}$	$1.16\ 10^{4}$	16 ± 3 (2s)	19 ± 6 (2s)	9 ± 3 (62s)	9 ± 2 (78s)
log-3	1413	29 445	$2.79\ 10^{11}$	$4.96\ 10^{3}$	19 ± 4 (1s)	21 ± 4 (1s)	9 ± 3 (12s)	8 ± 3 (12s)
log-4	2303	42529	$2.34\ 10^{28}$	$9.47\ 10^{4}$	77 ± 11 (2s)	94 ± 27 (2s)	10 ± 4 (72s)	11 ± 3 (81s)
log-5	2701	29483	$7.24\ 10^{38}$	$4.62\ 10^{5}$	121 ± 33 (18s)	147 ± 52 (17s)	-	-
tire-1	352	1022	$8.29\ 10^{36}$	$1.37\ 10^{3}$	74 ± 5 (<1s)	67 ± 7 (<1s)	3 ± 2 (7s)	3 ± 2 (8s)
tire-2	550	1980	$7.39\ 10^{11}$	$7.26\ 10^{2}$	20 ± 2 (<1s)	24 ± 4 (<1s)	5 ± 2 (3s)	5 ± 1 (3s)
tire-3	577	1984	$2.23\ 10^{11}$	$6.31\ 10^{3}$	20 ± 5 (1s)	19 ± 7 (1s)	6 ± 2 (18s)	6 ± 2 (20s)
tire-4	812	3197	$1.03\ 10^{14}$	$3.97\ 10^{3}$	27 ± 3 (1s)	28 ± 3 (1s)	8 ± 3 (7s)	7 ± 3 (10s)

TABLE 3 – Résultats expérimentaux pour les différentes stratégies sur des instances SAT codées par des circuits dDNNF.

d'échantillonnage géométrique pour estimer les vecteurs de perte [27]. À nouveau, cette méthode itérative peut être implémentée en temps linéaire (par itération) en utilisant la proposition 1. L'adaptation de OSMD aux semi-bandits est cependant moins immédiate : même si l'extension de CH atteint encore un regret espéré optimal dans ce contexte, son utilisation pratique reste limitée à cause de la projection de Bregman. Une question ouverte intéressante ici est de trouver si une combinaison de CH avec PCG est capable, en mode semi-bandit, d'atteindre un regret quasi-optimal en temps quadratique. Naturellement, le mode *bandit* pur est encore plus difficile. Sur ce point, Sakaue et. al. [32] ont obtenu de premiers résultats en utilisant des OBDDs pour une implémentation efficace de l'algorithme COMBBAND [7]. Mis à part ce résultat très récent, tout reste à faire dans les bandits combinatoires, notamment avec des circuits dDNNF, qui sont plus succincts que les OBDDs.

Références

[1] J-Y. Audibert, S. Bubeck, and G. Lugosi. Minimax policies for combinatorial prediction games. In *Proceedings of the 24th Annual Conference on Learning Theory (COLT 2011)*, pages 107–132, 2011.

[2] J-Y. Audibert, S. Bubeck, and G. Lugosi. Regret in online combinatorial optimization. *Mathematics of Operations Research*, 39(1) :31–45, 2014.

[3] M. A. Bashiri and X. Zhang. Decomposition-invariant conditional gradient for general polytopes with line search. In *Advances in Neural Information Processing Systems 30, (NIPS 2017)*, pages 2687–2697, 2017.

[4] A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operational Research Letters*, 31(3) :167–175, 2003.

[5] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2009.

[6] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[7] N. Cesa-Bianchi and G. Lugosi. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5) :1404–1422, 2012.

[8] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classification of Boolean Constraint Satisfaction Problems*. SIAM Monographs on Discrete Mathematics and Applications, 2001.

[9] A. Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4) :608–647, 2001.

[10] A. Darwiche. A compiler for deterministic, decomposable negation normal form. In *Proceedings of the 18th National Conference on Artificial Intelligence, (AAAI 2002)*, pages 627–634, 2002.

[11] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17 :229–264, 2002.

[12] M. E. Dyer, L. A. Goldberg, and M. Jerrum. The complexity of weighted Boolean #CSP. *SIAM Journal of Computing*, 38(5) :1970–1986, 2009.

[13] A. L. Friesen and P. M. Domingos. The sum-product theorem : A foundation for learning tractable models. In *Proceedings of the 33nd International Conference on Machine Learning, (ICML 2016)*, pages 1909–1918, 2016.

[14] D. Garber and O. Meshi. Linear-memory and decomposition-invariant linearly convergent conditional gradient algorithm for structured polytopes. In *Advances in Neural Information Processing Systems 29, (NIPS 2016)*, pages 1001–1009, 2016.

[15] J. Hannan. Approximation to Bayes risk in repeated play. *Contributions to the Theory of Games*, 3 :97–139, 1957.

[16] E. Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4) :157–325, 2016.

[17] D. P. Helmbold and M. K. Warmuth. Learning permutations with exponential weights. *Journal of Machine Learning Research*, 10 :1705–1736, 2009.

[18] M. Hutter and J. Poland. Adaptive online prediction by following the perturbed leader. *Journal of Machine Learning Research*, 6 :639–660, 2005.

[19] A. T. Kalai and S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3) :291–307, 2005.

[20] W. M. Koolen, M. K. Warmuth, and J. Kivinen. Hedging structured concepts. In *Proceedings of the 23rd Conference on Learning Theory (COLT 2010)*, pages 93–105, 2010.

[21] W. Krichene, S. Krichene, and A. M. Bayen. Efficient bregman projections onto the simplex. In *Proceedings of the 54th IEEE Conference on Decision and Control, (CDC 2015)*, pages 3291–3298, 2015.

[22] S. Lacoste-Julien and M. Jaggi. On the global linear convergence of frank-wolfe optimization variants. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 496–504, 2015.

[23] J-M. Lagniez and P. Marquis. An improved decision-dnnf compiler. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, (IJCAI 2017)*, pages 667–673, 2017.

[24] Y. T. Lee, A. Sidford, and S. C. Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science, (FOCS 2015)*, pages 1049–1065, 2015.

[25] M. Mohri. General algebraic frameworks and algorithms for shortest-distance problems. Technical Report 981210-10TM, AT & T Labs-Research, 1998.

[26] A. S Nemirovski and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. J. Wiley and Sons, 1983.

[27] G. Neu and G. Bartók. Importance weighting without importance weights : An efficient algorithm for combinatorial semi-bandits. *Journal of Machine Learning Research*, 17 :154 :1–154 :21, 2016.

[28] H. Poon and P. M. Domingos. Sum-product networks : A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pages 337–346, 2011.

[29] H. Rahmanian and M. K. Warmuth. Online dynamic programming. In *Advances in Neural Information Processing Systems 30, (NIPS 2017)*, pages 2824–2834, 2017.

[30] A. Rajkumar and S. Agarwal. Online decision-making in general combinatorial spaces. In *Advances in Neural Information Processing Systems 27, (NIPS 2014)*, pages 3482–3490, 2014.

[31] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3) :400–407, 1951.

[32] S. Sakaue, M. Ishihata, and S-I Minato. Efficient bandit combinatorial optimization algorithm with zero-suppressed binary decision diagrams. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, 2018.

[33] D. Suehiro, K. Hatano, S. Kijima, E. Takimoto, and K. Nagano. Online prediction under submodular constraints. In *Proceedings of the 23rd International Conference on Algorithmic Learning Theory (ALT 2012)*, pages 260–274, 2012.

[34] E. Takimoto and K. Hatano. Efficient algorithms for combinatorial online prediction. In *Proceedings of the- 24th International Conference on Algorithmic Learning Theory (ALT 2013)*, pages 22–32, 2013.

[35] E. Takimoto and M. K. Warmuth. Path kernels and multiplicative updates. *Journal of Machine Learning Research*, 4 :773–818, 2003.

GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms

Cédric Colas[1] Olivier Sigaud[1,2] Pierre-Yves Oudeyer[1]

[1] INRIA, Flowers Team, Bordeaux, France
[2] Sorbonne Université, ISIR, Paris, France

cedric.colas@inria.fr

Abstract

In continuous action domains, standard deep reinforcement learning algorithms like DDPG suffer from inefficient exploration when facing sparse or deceptive reward problems. Conversely, evolutionary and developmental methods focusing on exploration like novelty search, quality-diversity or goal exploration processes explore more robustly but are less sample efficient during exploitation. In this paper, we present the GEP-PG approach, taking the best of both worlds by sequentially combining two variants of a goal exploration process and two variants of DDPG. We study the learning performance of these components and their combination on a low dimensional deceptive reward problem and on the larger Half-Cheetah benchmark. We show that DDPG fails on the former and that GEP-PG obtains performance above the state-of-the-art on the latter.

This paper has been accepted at ICML 2018 and is available at https://arxiv.org/pdf/1802.05054.pdf.

Reconstruction d'état caché avec cartes auto-organisatrices récurrentes.

A. Dutech[1] J. Fix[2] H. Frezza-Buet[2]
[1] Université de Lorraine, CNRS, Inria, LORIA ; F-54000 Nancy, France
[2] Centrale-Supélec, LORIA ; F-57070 Metz, France
contact : alain.dutech@loria.fr

Mots Clef

Reconstruction d'état, Chaînes de Markov Cachées, Cartes auto-organisatrices

1 Motivations

Quand les états d'un processus ne sont pas Markoviens (POMDP par exemple), la convergence des algorithmes d'apprentissage par renforcement n'est pas garantie. Une solution est de reconstruire un processus Markovien en partant de la séquence des états. Dans ce but, nous explorons les capacités d'architectures récurrentes qui s'appuient sur des cartes neuronales auto-organisatrices pour apprendre à prédire des séquences d'observations issues de HMM.

Les algorithmes classiques d'apprentissage par renforcement [10] offrent des garanties de convergence quand ils sont appliqués à des problèmes qui peuvent se modéliser comme des Processus Décisionnels de Markov [8]. Or, dans de nombreux cas, la séquence d'information dont dispose l'agent pour apprendre n'est pas un processus markovien, l'agent n'a pas accès au *véritable* état du système (au sens de la physique ou de l'automatique), il n'en a qu'une observation partielle, bruitée, bien incomplète.

Si on se place dans le cadre formel des POMDP ([1, 2]), une manière de procéder est de considérer que cet état d'information est constitué des n dernières paires (o, a) d'observation et d'action. Dans le cas général, n doit être infini pour s'assurer que l'état d'information ainsi extrait est bien *complet*[1].

Nous voulons ici exploiter la puissance des réseaux de neurones récurrents pour extraire des états d'information les plus complets possibles dans le cadre de processus non-Markoviens. Mais contrairement à [6, 4, 7, 3] où ces états d'information sont appris indirectement, comme un moyen pour estimer la Q-fonction, nous voulons ici apprendre explicitement à extraire des états d'information. Pour cela, nous proposons une architecture neuronale récurrente qui s'appuie sur des ≪*Dynamic Self-Organizing Maps*≫ (DSOM) [9]. Les DSOM s'apparentent aux cartes auto-organisatrices de Kohonen qui sont connues pour leur bonnes propriétés dans le cadre de la quantification vectorielle [5], elles en diffèrent par une sensibilité réduite à la densité des échantillons d'apprentissage. Cette dernière propriété nous intéresse tout particulièrement dans le cadre général de l'apprentissage par renforcement. En effet, lors de l'apprentissage, il nous paraît pertinent d'accorder *a priori* autant d'importance aux régions de l'espace sensorimoteur visitées rarement qu'aux régions visitées souvent.

1. C'est-à-dire qu'il permet de contruire un Processus Décisionnel Markovien dont la solution est équivalente au POMDP original

2 Architecture

L'architecture neuronale récurrente que nous utilisons est décrite à la figure 1. En fonction de l'observation courante et de l'état actuel du réseau (le contexte), un neurone ≪vainqueur≫ est déterminé et l'état d'information est porté par ce neurone vainqueur, cela peut être son indice ou la valeur de son prototype.

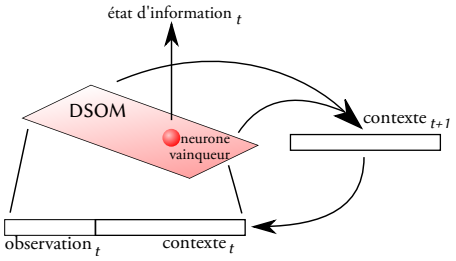


FIGURE 1 – **DSOM récurrent**.

3 Expérimentations

Dans un premier temps, nous avons vérifié que pour des HMM totalement observables (donc

Markoviens), l'architecture était convainquante. Par exemple, la figure 2 montre les états reconstruits pour un HMM qui produit la suite d'observation "`A-B-C-D-A-B-...`". Dans l'arc de cercle à droite, on positionne les derniers neurones vainqueur de cette carte mono-dimensionnelle par des petits ronds, la couleur des ronds est liée à l'observation de leur prototype et la flèche indique le contexte auquel ils réagissent. Les courbes montrent l'évolution de la distance du prototype vainqueur (observation en noir, contexte en bleu) et l'erreur de prédiction en observation. Ici, le réseau est capable d'apprendre la structure de la séquence (les ronds et flèches s'enchaînent bien).

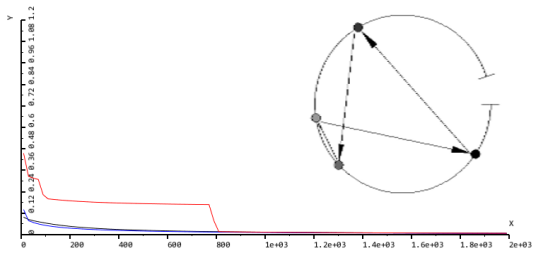


FIGURE 2 – **ABCD**.

Cela fonctionne aussi avec des HMM où il faut "compter" pour savoir quand l'observation change, comme par exemple avec des séquences du type "`A-A-A-A-A-F`", figure 3.

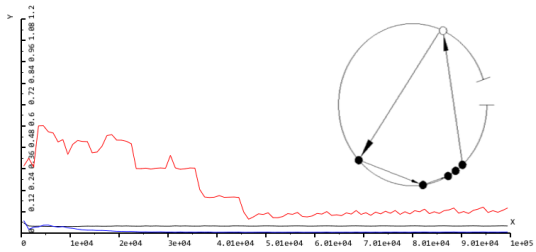


FIGURE 3 – **AAAAF**.

Mais les résultats sont – pour l'instant – décevants dans des cas comme "`A-B-C-B-A-...`", voir figure 4. Nous travaillons donc actuellement sur une meilleure différenciation des prototypes liés au contexte.

4 Références

Références

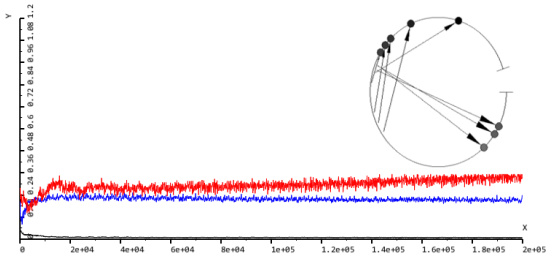


FIGURE 4 – **ABCBABCBA...**.

[1] ASTRÖM, K. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications 10* (1965), 174–205.

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

[3] DASWANI, M., SUNEHAG, P., HUTTER, M., ET AL. Feature reinforcement learning using looping suffix trees. In *10th European Workshop on Reinforcement Learning : JMLR : Workshop and Conference Proceedings 24* (2012), Journal of Machine Learning Research.

[4] DUTECH, A., AND SAMUELIDES, M. Apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés. *Revue d'Intelligence Artificielle (RIA) 17(4)* (2003), 559–589.

[5] KOHONEN, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43* (1982), 59–69.

[6] MCCALLUM, A. Learning to use selective attention and short-term memory in sequential tasks. In *From Animals to Animats, Proc. of the Fourth Int. Conf. on Simulating Adaptive Behavior* (1996).

[7] NGUYEN, P., SUNEHAG, P., AND HUTTER, M. Feature reinforcement learning in practice. In *European Workshop on Reinforcement Learning* (2011), Springer, pp. 66–77.

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

[9] ROUGIER, N. P., AND BONIFACE, Y. Dynamic Self-Organising Map. *Neurocomputing 74*, 11 (2011), 1840–1847.

[10] SUTTON, R. Generalization in reinforcement learning : Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8 (NIPS)* (1996), MIT Press, pp. 1038–1044.

5 Annexe : Architecture d'un RDSOM

L'architecture neuronale récurrente que nous utilisons est décrite à la figure 1. En fonction de l'observation courante et de l'état actuel du réseau (le contexte), un neurone *vainqueur* est déterminé et l'état d'information est porté par ce neurone vainqueur, cela peut être son indice ou la valeur de son prototype.
Plus formellement, les RDSOM que nous utilisons sont constitués d'un ensemble de N neurones qui sont chacun définis par :

- $i \in [0 \dots N)$: un indice
- $pos(i) \in [0, 1]$: une position dans l'espace des neurones (ici, un espace unidimensionnel, mais c'est arbitraire)
- $w(i) = (w_{in}(i), w_{rec}(i))$: un prototype où
  - $w_{in}(i) \in [0, 1]$: vecteur de l'espace des entrée $\mathcal{X}$ (ici de dimension 1, mais c'est arbitraire).
  - $w_{rec}(i) \in [0, 1]$: vecteur des poids récurrents dans l'espace des positions des neurones (et donc, ici, de dimension 1).

L'algorithme d'apprentissage se déroule en deux étapes.

Activation et détermination du neurone vainqueur A l'instant t, on présente au réseau une entrée $x_t = (o_t, c_t)$ composée d'une observation o_t et d'un contexte c_t. On calcule la *similarité* entre chaque neurone i et cette entrée, similarité qui se décompose en :

$$sim_{in}(o_t, i) = \exp\left(-\frac{{||o_t - w_{in}(i)||_2}^2}{2\sigma_{in}^2}\right), \quad (1)$$

$$sim_{rec}(c_t, i) = \exp\left(-\frac{{||c_t - w_{rec}(i)||_2}^2}{2\sigma_{rec}^2}\right). \quad (2)$$

Ces deux similarités sont combinées en une similarité globale

$$sim_g(x_t, i) = \sqrt{sim_{in}(o_t, i) \times \beta(1-\beta) sim_{rec}(c_t, i)} \quad (3)$$

où β est un réel de $[0; 1]$.
La similarité globale est elle-même convoluée [2] avec une gaussienne pour la lisser

$$sim(x_t, i) = \frac{1}{N} \sum_{k=-N/2}^{k=N/2} sim_g(x_t, \overline{i+k}) \exp\left(-\frac{k^2}{2\sigma^2}\right) \quad (4)$$

avec σ un paramètre à choisir et où $\overline{i+k} = (i+k)(modN)$, ce qui permet de déterminer le **neurone vainqueur** au temps t, dont l'indice est noté i_t^* :

$$i_t^* = \underset{i}{\operatorname{argmax}}\, sim(x_t, i). \quad (5)$$

On obtient aussi le **contexte** pour le pas de temps suivant en fonction de la position du neurone vainqueur.

$$c_{t+1} = pos(i_t^*) \quad (6)$$

Apprentissage des prototypes Le principe de l'apprentissage à l'instant t est de rapprocher les poids d'entrées et les poids récurents du couple (x_t, c_t). Chaque schéma d'apprentissage suit la même logique. On a donc pour tout neurone $j \in [0 \dots N)$:

$$w_{in}(j) \leftarrow w_{in}(j) + \epsilon \,.\, h(\nu_{in}, j, \underline{\mathrm{d}}(x_t, w_{in}(i_t^*)) \times \underline{\mathrm{d}}(x_t, w_{in}(j))\,(x_t - w_{in}(j)) \quad (7)$$

$$w_{rec}(j) \leftarrow w_{rec}(j) + \epsilon \,.\, h(\nu_{rec}, j, \underline{\mathrm{d}}(c_t, w_{rec}(i_t^*)) \times \underline{\mathrm{d}}(c_t, w_{rec}(j))\,(c_t - w_{rec}(j)) \quad (8)$$

avec

$$h(\nu, j, d) = \exp\left(-\frac{{||pos(i_t^*) - pos(j)||_2}^2}{\nu^2\, d^2}\right) \quad (9)$$

où ν, réél, est le paramètre d'élasticité de l'architecture.
Ici, on utilise des "distances normées", notées $\underline{\mathrm{d}}(.,.)$ qui sont toutes comprises entre 0 et 1.

$$\underline{\mathrm{d}}(x, y) = \frac{||x - y||_2}{\max_{x,y} ||x - y||_2} \quad (10)$$

Le noyau $h(\nu, j, d)$ permet de réguler la tendance des prototypes de chaque neurone j à se rapprocher de (x_t, c_t) en tenant compte de trois facteurs :

- plus le neurone vainqueur i_t^* est proche du prototype cible (x_t ou c_t), moins les autres neurones sont modifiés ;
- plus un neurone j est éloigné du vainqueur i_t^* (dans l'espaces des positions pos), moins il sera modifié ;
- enfin, plus le paramètre d'élasticité ν est petit, moins les neurones sont modifiés.

2. Pour cette convolution, nous considèrons que la similarité est un signal périodique, de période N.

APIA & CNIA & IC & JFPDA & RJCIA

IA pour l'éducation · IA & Santé · TAL & IA · Ethique & IA · Robotique & IA · France@IJCAI2018

PFIA 2018

11e Plate-forme Intelligence Artificielle

2 au 6 juillet 2018 - Nancy

Campus Sciences - Université de Lorraine, Vandoeuvre-lès-Nancy

JFPDA 2018

Actes des Journées Francophones Planification, Décision, Apprentissage pour la conduite des systèmes

Président du comité de programme: Olivier Buffet

PFIA2018.LORIA.FR

Recherche heuristique pour jeux stochastiques (à somme nulle)

Olivier buffet[1] · Jilles Dibangoye[2] · Abdallah Saffidine[3] · Vincent Thomas[1]

[1] Université de Lorraine, INRIA, CNRS, LORIA, F-54000 Nancy
[2] Université de Lyon, INSA Lyon and Inria, CITI, F-69000 Lyon
[3] Australian National University, Canberra, Australie

prenom.nom@{inria.fr|anu.edu.au}

Open Loop Execution of Tree-Search Algorithms

Erwan Lecarpentier[1] · Guillaume Infantes[1] · Charles Lesire[1] · Emmanuel Rachelson[2]

[1] DTIS (Traitement de l'Information et Systèmes), ONERA - The French Aerospace Lab, 2 avenue Edouard Belin, 31055 Toulouse, France
[2] DISC (Département d'Ingénierie des Systèmes Complexes), ISAE - Supaero, 10 avenue Edouard Belin, 31055 Toulouse, France

Génération de scénario : planification avec un opérateur défini par un modèle graphique

R. Lacaze-Labadie[1] · D. Lourdeaux[1] · M. Sallak[1]

[1] Sorbonne universités, Université de technologie de Compiègne, CNRS, Heudiasyc UMR 7253, 57 avenue de Landshut – 60203 COMPIEGNE Cedex, France

{remi.lacaze-labadie - domitile.lourdeaux - mohamed.sallak}@hds.utc.fr

Codage SMT dans un espace de plans (liens causaux) pour la planification temporelle en temps continu

Frédéric Maris · Maël Valais · Julien Vianey

IRIT, Université de Toulouse

{maris, mael.valais, julien.vianey}@irit.fr

Re-formation décentralisée d'équipes sous incertitude : modèle et propriétés structurelles

Jonathan Cohen · Abdel-Illah Mouaddib

Université de Caen Normandie

prénom.nom@unicaen.fr

Sur le Gradient de la Politique pour les Systèmes Multi-Agents Coopératifs

G. Bono[1] · J. Dibangoye[1] · L. Matignon[1,2] · F. Pereyron[3] · O. Simonin[1]

[1] Univ Lyon, INSA Lyon, INRIA, CITI, F-69621 Villeurbanne, France
[2] Univ Lyon, Université Lyon 1, LIRIS, CNRS, UMR5205, Villeurbanne, F-69622, France
[3] Volvo Group, Advanced Technology and Research

guillaume.bono@inria.fr

Learning to Act in Continuous Dec-POMDPs

Jilles S. Dibangoye[1] · Olivier Buffet[2]

[1] INSA de Lyon, laboratoire CITI, INRIA
[2] LORIA, INRIA

prénom.nom@inria.fr

Sur la Compilation de Jeux de Prédiction Combinatoire

Frédéric Koriche

CRIL, CNRS UMR 8188, Université d'Artois

frederic.koriche@cril.fr

GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms

Cédric Colas[1] · Olivier Sigaud[1,2] · Pierre-Yves Oudeyer[1]

[1] INRIA, Flowers Team, Bordeaux, France
[2] Sorbonne Université, ISIR, Paris, France

cedric.colas@inria.fr

Reconstruction d'état caché avec cartes auto-organisatrices récurrentes.

A. Dutech[1] · J. Fix[2] · H. Frezza-Buet[2]

[1] Université de Lorraine, CNRS, Inria, LORIA ; F-54000 Nancy, France
[2] CentraleSupélec, LORIA ; F-57070 Metz, France

contact : alain.dutech@loria.fr