Actes des 15es Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes

Frédéric Maris

AfIA

Association française
pour l'Intelligence Artificielle

JFPDA

Journées Francophones sur
la Planification, la Décision et l'Apprentissage
pour la conduite de systèmes

PFIA 2020

Table des matières

Éditorial

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.

Les journées précédentes ont eu lieu à Toulouse (2006), Grenoble (2007), Metz (2008), Paris (2009), Besançon (2010), Rouen (2011), Nancy (2012), Lille (2013), Liège (2014), Rennes (2015), Grenoble (2016), Caen (2017), Nancy (2018) et Toulouse (2019).

Cette année, dans un contexte sanitaire difficile, les 15es journées qui devaient avoir lieu à Angers, ont été vitualisées le 3 juillet 2020 dans le cadre de la Plate-forme Intelligence Artificielle.

Frédéric Maris

Comité de programme

Président

— Frédéric Maris (IRIT, Université Toulouse 3)

Membres

— Pegah Alizadeh (Pôle Léonard de Vinci)
— Aurélie Beynier (LIP6, Université Paris Sorbonne)
— Maroua Bouzid (GREYC, Université de Caen-Normandie)
— Olivier Buffet (INRIA, LORIA)
— Caroline Chanel (ISAE-SUPAERO)
— Martin Cooper (IRIT, Université Toulouse 3)
— Jilles Dibangoye (INSA Lyon)
— Alain Dutech (INRIA, LORIA)
— Humbert Fiorino (LIG)
— Laurent Jeanpierre (GREYC, Université de Caen-Normandie)
— Jérôme Lang (CNRS, LAMSADE, Université Paris-Dauphine)
— Laetitia Matignon (LIRIS CNRS, Université Lyon 1)
— Alexandre Niveau (GREYC, Université de Caen-Normandie)
— Cyril Pain-Barre (LIS, Université Aix-Marseille)
— Damien Pellier (LIG, Université Grenoble Alpes)
— Cédric Pralet (ONERA, Toulouse)
— Philippe Preux (INRIA, LIFL, Université de Lille)
— Emmanuel Rachelson (ISAE-SUPAERO)
— Pierre Régnier (IRIT, Université Toulouse 3)
— Régis Sabbadin (INRA)
— François Schwarzentruber (IRISA, ENS Rennes)
— Mathieu Serrurier (IRIT, Université Toulouse 3)
— Olivier Sigaud (ISIR, Université Paris Sorbonne)
— Florent Teichteil-Königsbuch (Airbus Central Research & Technology)
— Vincent Thomas (LORIA)
— Paul Weng (UM-SJTU Joint Institute)
— Bruno Zanuttini (GREYC, Université de Caen-Normandie)

Multiagent Planning and Learning As MILP

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

[1] Univ Lyon, INSA Lyon, INRIA, CITI, F-69621 Villeurbanne, France
[2] INRIA / Université de Lorraine, Nancy, France

prénom.nom@inria.fr

Résumé

Les processus décisionnels de Markov décentralisés et partiellement observables (Dec-POMDPs) offrent un cadre unifié pour la prise de décisions séquentielles par de plusieurs agents collaboratifs—mais ils restent difficiles à résoudre. Les reformulations en programmes linéaires mixtes (PLMs) se sont avérées utiles pour les processus décisionnels de Markov partiellement observables. Malheureusement, les applications existantes se limitent uniquement aux domaines mobilisant un ou deux agents. Dans cet article, nous exploitons une propriété de linéarisation qui nous permet de reformuler les contraintes non linéaires, omniprésentes dans les systèmes multi-agents, pour en faire des contraintes linéaires. Nous présentons en outre des approches de planification et d'apprentissage s'appuyant sur de nouvelles reformulations en PLMs des Dec-POMDPs, dans le cas général ainsi que quelques cas spécifiques. Les expérimentations sur des bancs de test standards à deux et plus de deux agents fournissent un solide soutien à cette méthodologie.

SMAs, Planification, Apprentissage, PLM

Abstract

The decentralized partially observable Markov decision process offers a unified framework for sequential decision-making by multiple collaborating agents but remains intractable. Mixed-integer linear formulations proved useful for partially observable domains, unfortunately existing applications restrict to domains with one or two agents. In this paper, we exploit a linearization property that allows us to reformulate nonlinear constraints from n-agent settings into linear ones. We further present planning and learning approaches relying on MILP formulations for general and special cases, including network-distributed and transition-independent problems. Experiments on standard 2-agent benchmarks as well as domains with a large number of agents provide strong empirical support to the methodology.

MAS, Planning, Learning, MILP

1 Introduction

The decentralized partially observable Markov decision process offers a unified framework to solving cooperative, decentralized stochastic control problems Bernstein et al. [2002]. This model encompasses a large range of real-world problems in which multiple agents collaborate to optimize a common objective. Central to this setting is the assumption that agents can neither see the actual state of the world nor explicitly communicate their observations with each other due to communication cost, latency or noise. This assumption partially explains the worst-case complexity: finite-horizon cases are in NEXP; and infinite-horizon cases are undecidable Bernstein et al. [2002]. A general methodology to solving decentralized stochastic control builds upon the concept of occupancy states, *i.e. sufficient statistics* for evaluating and selecting decentralized policies Dibangoye et al. [2013], Oliehoek [2013]. The occupancy-state space is a probability simplex of points in the Cartesian product over the state and history spaces. For every occupancy state, dynamic programming and reinforcement learning approaches compute and store tables consisting of one value per state-history pair. Unfortunately, the state space grows exponentially with the number of state variables, and the history space expands doubly exponentially with time. Known as the *curse of dimensionality*, these phenomena render existing approaches intractable in the face of decentralized decision-making problems of practical scale.

Methods that can overcome the curse of dimensionality previously arose in the literature of decentralized control, but restricting to 2-agent settings. Examples include memory-bounded dynamic programming Seuken and Zilberstein [2008], Kumar et al. [2015] and linear and nonlinear programming using finite-state controllers Amato et al. [2010], Kumar et al. [2016]. These successes shed light on approximate dynamic and linear programming as potentially powerful tools for large-scale decentralized partially observable Markov decision processes. One approach to dealing with the curse of dimensionality is to rely on parametrized occupancy measures Dibangoye et al.

[2013]. However, choosing a parametrization that can closely mimic the desired occupancy measures requires human expertise or theoretical analysis. Though crucial, the design of an approximation architecture goes beyond the scope of this paper. Instead this paper focusses on exact planning and learning approaches relying on MILP for computing good decentralized policy given parametrized occupancy measures.

To this end, we first exploit a linearization property that allows us to reformulate into linear ones all nonlinear constraints that arose from multiple collaborating agents. We then present a general MILP formulation for n-agent settings, restricting attention to deterministic finite-memory decentralized policies[1]. In addition, we introduce a two-phase approach that produces a sequence of decentralized policies and dynamics' models through iteration. At the first phase called model estimation, we maintain statistics about the dynamics' model. At the second phase namely policy improvement, we rely on our MILP formulation to calculate a new decentralized policy based on the current dynamics' model. Under discounted reward criterion, the sequence of decentralized policies converges to some optimal deterministic and finite-memory decentralized policy. We further demonstrate how to use this planning and learning scheme to exploit two properties present in many decentralized control problems, namely joint-full observability and weak separability Becker et al. [2004], Nair et al. [2005]. Experiments on standard 2-agent benchmarks as well as domains with a large number of agents provide strong empirical support to the methodology for planning and learning good decentralized policies.

2 Related Work

Mixed-integer linear programming was used previously for decentralized decision-making, but always with a focus different from ours. Much of the effort has been directed toward exact formulations for restricted classes of either problems or policies. Witwicki and Durfee [2007] and later on Mostafa and Lesser [2011] presented formulations for 2-agent transition independent decentralized Markov decision processes Becker et al. [2004]. Aras and Dutech [2010] introduced an exact formulation for 2-agent finite-horizon decentralized decision-making, which inevitably scales poorly with the number of state variables and planning horizon. More recently, Kumar et al. [2016] proposed yet another 2-agent formulation but with a focus on finite-state controllers. Unfortunately, there are a number of factors that may affect the performance of solvers while optimizing finite-state controllers. First, the numbers of variables and contraints grow linearly with the number of agents, states, actions, and observations; even more importantly they grow quadratically with the number of nodes per controller. Consequently, MILP formulations of interest for practical problems involve prohibitively large numbers of variables and constraints. The second limitation is somewhat imperceptible and has to do with the semantic of nodes in finite-state controllers. Each node aims at representing a partition of the history space as well as prescribing an action to be taken in that node. Interestingly, these separate decisions are interconnected. The actions to be taken in all nodes may affect the histories to be subsumed in each specific node and vice-versa. Taken all together, these barriers make optimizing finite-state controllers particularly hard tasks, which explains the impetus for the development of a novel approximation approach. Kumar et al. [2016] suggest a heuristic search method that can incrementally build small 2-agent finite-state controllers. However, to the best of our knowledge, no existing MILP formulation for decentralized decision-making can cope with problems with more than two agents. The primary contribution of this paper is to provide the first attempt to handle this issue.

To handle the adaptive case, *i.e.* when the dynamics model is unknown, a standard approach is reinforcement learning. Existing reinforcement learning methods for decentralized decision-making extend adaptive dynamic programming and policy search approaches from single to multiagent settings. Currently, no multiagent reinforcement learning methods is based on MILP formulations. This is somewhat surprising since, there are single-agent reinforcement learning methods based linear programming. One such approach, namely `probing`, consists of two phases: (i) the estimation phase, where the transition probabilities are updated; and (ii) the control phase, where a new policy policy is calculated based on the current transition probabilities Altman [1999]. The algorithm iterates these two phases forever or until the training budget is exhausted. In this paper, we extend this approach to decentralized decision-making, thus providing the first multiagent model-based reinforcement learning method based on MILP formulations.

3 Backgrounds

The paper makes use of the following notation. $\delta_y(x)$ is the Kronecker delta function. For any arbitrary finite set B, $|B|$ denotes the cardinality of B, $\mathbb{N}_{\leq |B|} = \{0, 1, \ldots, |B|\}$, $\mathbb{N}^+_{\leq |B|} = \{1, \ldots, |B|\}$, and $\triangle(B)$ is the $(|B| - 1)$-dimensional real simplex. Also, we use shorthand notations $b_{1:n} = (b_1, b_2, \ldots, b_n)$ and $b^\top$ to denote the transpose of b.

3.1 Problem Formulation

A decentralized partially observable Markov decision process is given by a tuple $M_n \doteq (X, U, Z, p, q, r, \nu)$ made of: a finite set of n agents; a finite state space X; a finite action space $U = U_1 \times U_2 \times \cdots \times U_n$; a finite observation space $Z = Z_1 \times Z_2 \times \cdots \times Z_n$; state transition probabilities $p^u(x, y)$ representing, for each pair (x, y) of states and each action u, the probability that next state will be y given

[1] Though randomized finite-memory decentralized policies should achieve better performances than deterministic ones, the corresponding optimization problem is non-convex, often leading to poorer or equivalent random solutions in comparison to deterministic ones Amato et al. [2010].

that the current state is x and the current action is u; observation probabilities $q^u(y, z)$ representing, for each state y, each action u, and each observation z, the probability that next observation will be z after taking action u is leading to state y; rewards $r(x, u)$ representing, for each state x and action u, the reward incurred when taking action u in current state x; and ν is the initial state-history distribution.
Solving decentralized stochastic control problems aims at finding a (decentralized) policy a, *i.e.,* n independent policies $(a_1, a_2, \ldots, a_n)$, one individual policy for each agent. Each policy a prescribes actions conditional on (action-observation) histories $o = (o_1, o_2, \ldots, o_n)$, initially $o \doteq (\emptyset, \emptyset, \ldots, \emptyset)$, such that

$$a(o, u) = \prod_{i=1}^{n} a_i(o_i, u_i), \qquad \forall o \in O, u \in U, \qquad (1)$$

where $O = O_1 \times \cdots \times O_n$ is a finite set of histories, ranging from 0- to ℓ-steps histories. We shall restrict attention to ℓ-order Markov policies. These policies map 0- to ℓ-steps histories to actions, in particular 1-order Markov policies are called Markov policies. For any arbitrary ϵ, one can choose $\ell = \log_\alpha(1-\alpha)\epsilon/\|r\|_\infty$ so that there always exists at least one ℓ-order policy within ϵ of an optimal one. For each policy a, we define a transition matrix P^a, where each entry $P^a(x, o, x', o')$ denotes the probability of transiting from state-history (x, o) to state-history (x', o') under policy a.
Policies of interest are those that achieve the highest performance. In this paper, we consider the infinite-horizon normalized discounted reward criterion, which ranks policies a according to the initial state-history distribution ν and a discount factor $\alpha \in (0, 1)$ as follows:

$$J_\alpha(a; \nu) \doteq (1-\alpha)\textstyle\sum_{\tau=0}^{\infty} \alpha^\tau \mathbb{E}^a_\nu\{r(x_\tau, u_\tau)\}, \qquad (2)$$

where $\mathbb{E}^a_\nu\{\cdot\}$ denotes the expectation with respect to state-action-history distributions $P^a_\nu(\tau)$ at each time step τ conditional on distribution ν and policy a, also known as a τ-th *occupancy state*. $P^a_\nu(\tau; x_\tau, o_\tau)$ denotes the probability of being in state x_τ after experiencing history o_τ at decision epoch τ when agents follow policy a starting in ν. An optimal policy $a^* \in \arg\max_a J_\alpha(a; \nu)$ is one that achieves the unique optimal value $J_\alpha(\nu) = J_\alpha(a^*; \nu)$.

3.2 Occupancy Measure

To overcome the fact that agents can neither see the state of the world nor explicitly communicate with one another, Szer et al. [2005] suggest formalizing decentralized stochastic control problems from the perspective of a centralized offline algorithm. A centralized algorithm selects, for each decision epoch, and based on the corresponding occupancy state, a decentralized rule to be executed by the agents. In general, resulting policies are non-stationary, *i.e.* rules at two consecutive decision epochs may be different. In this paper, however, we restrict rules at each decision epoch to be identical, thus we focus on stationary decentralized policies. This choice gives rise to statistics, namely *occupancy measures* $s_\alpha(\nu, a)$, that merge together occupancy states $\{P^a_\nu(\tau)\}_{\tau \in \mathbb{N}}$ encountered under policy a starting in ν while preserving ability to estimate the α-discount reward $J_\alpha(a; \nu)$:

$$s_\alpha(\nu, a) \doteq (1-\alpha) \sum_{\tau=0}^{\infty} \alpha^\tau P^a_\nu(\tau). \qquad (3)$$

Interestingly, the occupancy measure comes with many important properties.

Lemma 1. *$s_\alpha(\nu, a)$ is a probability distribution.*

Proof. Let $\boldsymbol{e}$ be a vector of all ones. Then we have

$$\begin{aligned}
&s_\alpha(\nu, a)^\top e \\
&\doteq \left((1-\alpha)\nu^\top \sum_{\tau=0}^{\infty} \alpha^\tau (P^a)^\tau\right)\boldsymbol{e} \\
&= (1-\alpha)\nu^\top \sum_{\tau=0}^{\infty} \alpha^\tau \boldsymbol{e} \\
&= (1-\alpha)\nu^\top (1-\alpha)^{-1}\boldsymbol{e}, \text{ given } \sum_{\tau=0}^{\infty} \alpha^\tau = (1-\alpha)^{-1} \\
&= \nu^\top \boldsymbol{e} \\
&= 1, \ \nu \text{ being a probability distribution}
\end{aligned}$$

and the claim follows. □

If α were seen as a survival probability at each time step, then $s_\alpha(\nu, a)$ gives, for state-history pair (x, o) the probability to be in that situation just before dying. Combining (2) and (3), it appears that $J_\alpha(a; \nu)$ is a linear function of occupancy measure $s_\alpha(\nu, a)$:

Lemma 2. *$s_\alpha(\nu, a)$ is a sufficient statistic for estimating infinite-horizon normalized discounted reward:*

$$J_\alpha(a; \nu) = \mathbb{E}^a_{s_\alpha(\nu,a)}\{r(x, u)\}. \qquad (4)$$

Proof. Let measure $\nu^\top (P^a)^\tau$ be the probability distribution over state-history pairs conditional on initial state distribution ν and policy a, after τ decision epochs. Then we

have

$$
\begin{aligned}
&J_\alpha(a;\nu)\\
&\doteq (1-\alpha)\sum_{\tau=0}^{\infty}\alpha^\tau E^a_\nu\{r(x_\tau,u_\tau)\}\\
&=(1-\alpha)\sum_{\tau=0}^{\infty}\alpha^\tau E\{r(x_\tau,u_\tau)\mid\\
&\{(x_\tau,o_\tau)\sim\nu^\top(P^a)^\tau,u_\tau\sim a(o_\tau,\cdot)\}\\
&=(1-\alpha)\sum_{\tau=0}^{\infty}\alpha^\tau\\
&\sum_{x\in X}\sum_{o\in O}(\nu^\top(P^a)^\tau)(x_\tau=x,o_\tau=o)\sum_{u\in U}a(o,u)\cdot r(x,u)\\
&=\sum_{x\in X}\sum_{u\in U}r(x,u)\\
&\sum_{o\in O}a(o,u)\left((1-\alpha)\nu^\top\sum_{\tau=0}^{\infty}\alpha^\tau(P^a)^\tau\right)(x_\tau=x,o_\tau=o)\\
&=\sum_{x\in X}\sum_{u\in U}r(x,u)\sum_{o\in O}a(o,u)\cdot s_\alpha(\nu,a;x,o)\\
&\doteq E^a_{s_\alpha(\nu,a)}\{r(x,u)\}
\end{aligned}
$$

and the claim follows. □

If we let I be the identity matrix, we have:

$$\sum_{\tau=0}^{\infty}\alpha^\tau P^a_\nu(\tau)=\nu^\top\sum_{\tau=0}^{\infty}\alpha^\tau(P^a)^\tau\ =\nu^\top(I-\alpha P^a)^{-1} \tag{5}$$

Injecting (5) into (3), and re-arranging terms lead to a recursive characterization of occupancy measures:

$$s_\alpha(\nu,a)^\top(I-\alpha P^a)=(1-\alpha)\nu^\top. \tag{6}$$

To solve M_n, it will prove useful to search both a policy a and the corresponding occupancy measure $s_\alpha(\nu,a)$. Therefore, we define the *extended occupancy measure* $\zeta_\alpha(\nu,a)\ =\ \{\zeta_\alpha(\nu,a;x,o,u)\}$ over state-history-action triplets, associated with each policy a, initial distribution ν, and discount factor α, and given by

$$\zeta_\alpha(\nu,a;x,o,u)\doteq s_\alpha(\nu,a;x,o)\cdot a(o,u),\quad\forall x,o,u. \tag{7}$$

This measure captures the frequency of visits of each state-history-action triplet when the system runs under policy a, conditioned on initial distribution ν.

4 MILP Reformulations

To motivate the role of extended occupancy measures (7), let us start with a mathematical program to finding a^*. Consider problem $(\mathcal{P}_1)$ given by:

$$\text{Maximize}_{a,a_{1:n},\zeta}\ \mathbb{E}_s\{r(x,u)\}\text{ subject to: (1) and (6)}$$

where a_i is agent i's policy, a defines the decentralized policy, and ζ denotes the extended occupancy measure. It can be shown, using (6), that any feasible ζ of $(\mathcal{P}_1)$ is an extended occupancy measure $\zeta_\alpha(\nu,a)$ under policy a. It follows that, for any $\zeta=\zeta_\alpha(\nu,a)$ solution of program $(\mathcal{P}_1)$, policy a is optimal for any selected class of finite-memory policies. Unfortunately, $(\mathcal{P}_1)$ is a nonlinear optimization problem, with many local optima. Earlier attempts to solving $(\mathcal{P}_1)$—for one or two agents only—make use of nonlinear solvers, often leading to local optima Amato et al. [2010]. The remainder of this section presents an exact mixed-integer linear program for M_n, restricting attention to deterministic and stationary ℓ-order Markov policies, as they have been shown to achieve ϵ-optimal performance.

4.1 ℓ-order Markov Policies

Notice that (6) is a nonlinear constraint, so that $(\mathcal{P}_1)$ is not a MILP. Therefore, finding an ϵ-optimal policy by directly solving $(\mathcal{P}_1)$ is hopeless in general, though from case to case nonlinear programming may achieve good results Amato et al. [2010]. However, it is possible to reformulate the constraints to transform the problem into a MILP. Previous linearization of nonlinear programs to solving decentralized stochastic control problems have been limited to two-agent cases, with either specific problem assumptions, *e.g.,* transition-independent settings Wu and Durfee [2006], or restricted classes of policies, *e.g.,* sequence-form policies Aras and Dutech [2010]; finite-state controllers Kumar et al. [2016].

Next, we introduce a mixed-integer linear programming approach for general discrete-time decentralized stochastic control problems. Before proceeding any further let us provide preliminary properties that will be useful to establish the main results of the paper. In particular, we present a linearization property that allows us to formulate nonlinear constraints in $(\mathcal{P}_1)$ as linear constraints. We start with the linearization of the product between Boolean and continuous variables Berthold et al. [2009].

Lemma 3 (Berthold et al. [2009])**.** *If we let v_1, v_2, and w be Boolean, random, and non-negative variables, respectively; and*

$$(C_1)\left|\begin{array}{ll} w-v_k & \leq 0\quad\forall k\in\{1,2\}\\ v_1+v_2-w & \leq 1\end{array}\right.$$

then solutions of polyhedron (C_1) satisfy $w=v_1\cdot v_2$.

Proof. Building upon Berthold et al. [2009], the proof proceeds by considering all possible values for v_1.

1. If $v_1=0$, then from $w-v_k\leq 0$ and $w\in[0,1]$, we know that $w=0$ no matter v_2, which satisfies $w=v_1\cdot v_2$.
2. On the other hand, if $v_1=1$, then from $v_1+v_2-w\leq 1$ and $w\in[0,1]$, we know that $w\leq 1$ for $v_1=v_2$, which is further tightened when considering $v_1=v_2$, *i.e.,* $w\leq v_2$. The last inequality $v_1+v_2-w\leq 1$

shows that $w \geq v_2$. As a consequence $w = v_2$, which satisfies $w = v_1 \cdot v_2$.

□

The next property shows for the first time how to exploit Lemma 3 to reformulate nonlinear constraints from n-agent cases into linear ones.

Proposition 1. *If we let $\zeta(x, o_{1:n}, u_{1:n})$ be a joint distribution and $\{a_i(o_i, u_i)\}_{i \in \mathbb{N}^+_{\leq n}}$ be Boolean variables; and*

$$(C_\zeta(o_i, u_i)) \left| \begin{array}{ll} P_\zeta(o_i, u_i) - a_i(o_i, u_i) & \leq 0 \\ P_\zeta(o_i) + a_i(o_i, u_i) - P_\zeta(o_i, u_i) & \leq 1 \end{array} \right.$$

then solutions of polyhedron $\{C_\zeta(o_i, u_i)\}_{i \in \mathbb{N}^+_{\leq n}}$ satisfy

$$\begin{array}{ll} \zeta(x, o_{1:n}, u_{1:n}) & = P_\zeta(x, o_{1:n}) \prod_{i=1}^n P_\zeta(u_i|o_i) \\ a_i(o_i, u_i) & = P_\zeta(u_i|o_i). \end{array}$$

Proof. The extended occupancy state $\zeta(x, o, u)$ can be rewritten equivalently as follows:

$$\begin{aligned} \zeta(x, o, u) &\doteq P_\zeta(x, o_{1:n}) \prod_{j=1}^n P_\zeta(u_j|o_j) \\ &= P_\zeta(x, o_{-i}, u_{-i}|o_i, u_i) P_\zeta(o_i) P_\zeta(u_i|o_i) \quad (8) \\ &= P_\zeta(x, o_{-i}, u_{-i}|o_i, u_i) P_\zeta(o_i, u_i) \quad (9) \end{aligned}$$

Since LHS of both (8) and (9) are equal, we have equivalently

$$P_\zeta(o_i, u_i) = P_\zeta(o_i) P_\zeta(u_i|o_i), \qquad \forall i, o_i, u_i. \quad (10)$$

Using Lemma 3 along with the fact that $\{P_\zeta(u_i|o_i)\}$ are Boolean variables (and the obvious result that $P_\zeta(o_i, u_i) - P_\zeta(o_i) \leq 0$), we know that solutions of $\{C_\zeta(o_i, u_i)\}$ also satisfy (10). Equality $a_i(o_i, u_i) = P_\zeta(u_i|o_i)$ follows directly from Lemma 3. Indeed, if $P_\zeta(o_i) = 0$, the first inequality implies $a_i(o_i, u_i) = 0 (= P_\zeta(u_i|o_i))$; otherwise $P_\zeta(o_i) = 0$, and Lemma 3 gives us $P_\zeta(o_i, u_i) = a_i(o_i, u_i) \cdot P_\zeta(o_i) \neq 0$, so that $a_i(o_i, u_i) = P_\zeta(o_i, u_i)/P_\zeta(o_i) = P_\zeta(u_i|o_i)$ using Bayes rule. □

We are now poised to present a MILP to solving general decentralized stochastic control problems.

Theorem 1. *If we let $\{a_i(o_i, u_i)\}$ and $\{\zeta(x, o, u)\}$ be Boolean and non-negative variables, respectively, then a solution of mixed-integer linear program $(\mathcal{P}_2)$:*

$$\max_{a_{1:n}, \zeta} \mathbb{E}_\zeta\{r(x, u)\} \text{ s.t. (1) and } \{C_\zeta(o_i, u_i)\}_{i, o_i \in O_i, u_i \in U_i}$$

is an ϵ-optimal solution of $(\mathcal{P}_1)$, where a_i is agent i's policy and ζ denotes the extended occupancy measure.

Proof. From Oliehoek et al. [2008], we know there always exists a deterministic history-dependent decentralized policy that is as good as any randomized history-dependent decentralized policy. Moreover, as previously discussed, by restricting attention to ℓ-order Markov decentralized policies, the best possible performance in this subclass is within $\|r\|_\infty \cdot \alpha^\ell/(1-\alpha)$ of the optimal performance, *i.e.,* the regret of taking arbitrary decisions from time step ℓ onward. Hence, by searching in the space of deterministic and ℓ-order Markov decentralized policies, *i.e.,* one individual ℓ-order Markov policy a_i for each agent $i \in \mathbb{N}^+_{\leq n}$, we preserve ability to find an ϵ-optimal solution of the original problem M_n under the discounted-reward criterion, where $\epsilon \leq \|r\|_\infty \cdot \alpha^\ell/(1-\alpha)$. Since a_i is agent i's policy and ζ denotes the extended occupancy measure, we know that $\{a_i(o_i, u_i)\}$ and $\{\zeta(x, o_{1:n}, u_{1:n})\}$ are Boolean and non-negative variables, respectively. Thus, from Proposition 1, we have that solutions of polyhedron $\{C_\zeta(o_i, u_i)\}_{i, o_i \in O_i, u_i \in U_i}$ satisfy $\zeta(x, o_{1:n}, u_{1:n}) = s(x, o_{1:n}) \prod_{i=1}^n a_i(o_i, u_i)$ and $s(x, o_{1:n})$ are marginal probabilities $P_\zeta(x, o_{1:n})$ and $a_i(o_i, u_i)$ are conditional probabilities $P_\zeta(u_i|o_i)$ for $i \in \mathbb{N}^+_{\leq n}$. □

This theorem establishes a general MILP to finding an ϵ-optimal policy in M_n under the discounted-reward criterion. We will refer to this problem as the exact MILP. Unfortunately, the state, action, history spaces for practical problems are enormous due to the curse of dimensionality. Consequently, the MILP of interest involves prohibitively large numbers of variables and constraints. $(\mathcal{P}_2)$ considers less constraints than its nonlinear counterpart $(\mathcal{P}_1)$, but the same number of variables. Variables in $(\mathcal{P}_1)$ are all free, whereas variables $\{a_i(o_i, u_i)\}_{i \in \mathbb{N}^+_{\leq n}, o_i \in O_i, u_i \in U_i}$ in $(\mathcal{P}_2)$ are Boolean and remainders $\{\zeta(x, o, u)\}_{x \in X, o \in O, u \in U}$ are free.

5 Tractable Subclasses

In this section, we present two examples involving the mixed-integer linear formulations for subclasses of decentralized partially observable Markov decision processes. The intention is to illustrate more concretely how the formulation might be achieved and how reasonable choices lead to near-optimal policies. We shall consider *joint-observability* and *weak-separability* assumptions.

5.1 Joint observability assumption

We first consider a setting where agents collectively observe the true state of the world. This assumption, known as joint observability, arises in many decentralized Markov decision processes Bernstein et al. [2002], *e.g.,* transition-independent decentralized Markov decision processes Becker et al. [2004]. More formally, we say that a system is jointly observable if and only if there exists a surjective function $\varphi\colon Z \mapsto X$ which prescribes the true state of the world given the current joint observation.

Corollary 1. *Under joint observability assumption, if we let $a_i(z_i, u_i)_{i \in \mathbb{N}^+_{\leq n}}$ and $\zeta(z, u)$ be Boolean and non-negative variables, respectively, then:*
(i) the transition probability from observation z to obser-

vation z' upon taking action u is

$$p^u_\varphi(z, z') \doteq \sum_{x\in X}\delta_x(\varphi(z))\sum_{y\in X}p^u(x, y) \cdot q^u(y, z'),$$

where the rewards over observations is given by $r_\varphi(z, u) \doteq \sum_{x\in X}\delta_x(\varphi(z)) \cdot r(x, u)$; and the initial distribution over observations is given by $\nu_\varphi(z) \doteq \sum_{x\in X}\delta_x(\varphi(z))\nu(x)$; (ii) the occupancy measures over observations satisfy

$$s^\top(I - \alpha P^a) \doteq (1 - \alpha)\nu^\top_\varphi; \tag{11}$$

(iii) a solution of mixed-integer linear program $(\mathcal{P}_3)$

$$\max_{a_{1:n},\zeta} \mathbb{E}_\zeta\{r_\varphi(z, u)\} \text{ s.t. (11) and } \{C_\zeta(z_i, u_i)\}_{i,z_i\in Z_i,u_i\in U_i}$$

is also solution of $(\mathcal{P}_2)$, where a_i is agent i's policy and ζ denotes the extended occupancy measure.

This corollary presents an approximate mixed-integer linear program that can find Markov decentralized policies under joint observability. Markov policies, a.k.a. 1-order Markov policies, act depending only upon the current observation. This formulation depends on states and histories only through the current observations, which results in a significant reduction in the number of variables and constraints, *i.e.,* from $\mathbf{O}(|X||O||U|)$ in $(\mathcal{P}_2)$ to $\mathbf{O}(|Z||U|)$ in $(\mathcal{P}_3)$. Interestingly, this formulation finds optimal policies in transition-independent decentralized Markov decision processes, as deterministic Markov policies were proven to be optimal in such a setting Goldman and Zilberstein [2004].

5.2 Weak separability assumption

Next, we consider the weak separability assumption, which arises in network-distributed partially observable Markov decision processes Nair et al. [2005]. The assumption allows us to decouple variables involved in the approximate mixed-integer linear programs into factors, *i.e.,* subsets of variables, which make it possible to scale up to large number of agents. The intuition behind this assumption is that not all agents interact with one another; often an agent interacts only with a small subset of its neighbors, hence its decisions may not affect the remainder of its teammates. To take into account the locality of interaction, we make the following assumptions.

Definition 1. *Let E be a set of subsets e of agents. A decentralized partially observable Markov decision process $(n, X, Z, U, q, p, r, \nu)$ is said to be weakly separable if the following holds: n denotes the number of agents; $X \doteq X_0 \times X_1 \times \cdots \times X_n$; ν is multiplicatively fully separable,* i.e., *there exists $(\nu_i)_{i\in\mathbb{N}_{\leq n}}$ such that $\nu(x) = \prod_{i\in\mathbb{N}_{\leq n}}\nu_i(x_i)$, where $x = (x_i)_{i\in\mathbb{N}_{\leq n}}$; p is multiplicatively weakly separable,* i.e., *there exists $(p_i)_{i\in\mathbb{N}_{\leq n}}$ such that $p^{u_e}_e(x_e, y_e) = p_0(x_0, y_0)\prod_{i\in e}p^{u_i}_i(x_i, y_i)$, where $x_e = (x_0, (x_i)_{i\in e})$ and $u_e = (u_i)_{i\in e}$; q is multiplicatively weakly separable,* i.e., *there exists $(q_i)_{i\in\mathbb{N}^+_{\leq n}}$ such that $q^{u_e}_e(z_e, y_e) = \prod_{i\in e}q^{u_i}_i(z_i, y_i)$, where $y_e = (y_i)_{i\in e}$, $z_e = (z_i)_{i\in e}$ and $u_e = (u_i)_{i\in e}$; r is additively weakly separable,* i.e., *there exists $(r_e)_{e\in E}$ such that $r(x, u) = \sum_{e\in E}r_e(x_e, u_e)$, for all state and action x, u.*

This assumption suggests two agents, i and j, can only affect one another if they share the same subset e, *i.e.,* $i, j \in e$; otherwise they can choose what to do with no knowledge about what the other sees or plans to do. As a consequence, the value function in this setting is proven to be additively weakly separable Dibangoye et al. [2014a], *i.e.,* $\mathbb{E}_\zeta\{r(x, u)\} = \sum_{e\in E}\mathbb{E}_{\zeta_e}\{r_e(x, u)\}$, where

$$s^\top_e(I - \alpha P^{a_e}) \doteq (1 - \alpha)\nu^\top_e, \tag{12}$$

describes the recursion definition of the occupancy measure s_e extract from the extended occupancy measure ζ_e. The following exploits this property to define an exact mixed-integer linear program that decouples variables according to E, resulting in significant dimensionality reduction.

Corollary 2. *Let M_n be weakly separable. If we let $\{a_i(o_i, u_i)\}_{i\in\mathbb{N}^+_{\leq n}}$, and $\{\zeta_e(x_e, o_e, u_e)\}$ be Boolean and non-negative variables, respectively; then a solution of mixed-integer linear program $(\mathcal{P}_4)$*

$$\max_{a_{1:n},\zeta} \textstyle\sum_{e\in E}\mathbb{E}_{\zeta_e}\{r_e(x_e, u_e)\} \text{ s.t. (12) and } \{C_{\zeta_e}(o_i, u_i)\}$$

is also a network-distributed solution of $(\mathcal{P}_2)$, where where a_i is agent i's policy and ζ_e denotes the extended occupancy measure for all e.

This mixed-integer linear program exploits the so-called weak separability assumption that arises under locality of interaction. It is worth mentioning that $(\mathcal{P}_4)$ can find an optimal policy for network-distributed partially observable Markov decision processes, assuming reasonable choice of histories O_e for each subset of agents e Dibangoye et al. [2014a].

6 Adaptive Decentralized Control

In this section, we extend to decentralized partially observable Markov decision processes the probing algorithm originally introduced for Markov decision processes. Similarly to the original algorithm, ours alternates between model estimation and policy improvement phases forever or until the training budget is exhausted. The estimators of both dynamics $\{P_\tau\}$ and exploration policies $\{\pi_\tau\}$ shall involve counting the number of times the algorithm visits state-action-state-observation quadruplets (x, u, y, z), state-action pairs (x, u), and states x after τ interactions between the agent and the environment—by an abuse of notation we shall use $w_\tau(x, u, y, z)$, $w_\tau(x, u)$, $w_\tau(x)$ to store these numbers, respectively. It is worth noticing that since the model does not depend on histories, maintaining history-dependent policies $\{a_\tau\}$ is useless, instead it suffices to maintain state-dependent policies $\{\pi_\tau\}$ corresponding to extended occupancy measures $\{\zeta_\tau\}$.

Under standard ergodicity conditions, the model estimation phase ensures each state-action pairs is visited infinitely often, making $P_\tau = \{p_\tau^{u,z}(x,y)\}$ a consistent estimator of dynamics after τ interactions: $p_\tau^{u,z}(x,y) = w_\tau(x,u,y,z)/w_\tau(x,u)$, if $w_\tau(x,u) > 0$ and chosen arbitrary otherwise. In other terms, if each state-action pair is visited infinitely often, then by the strong law of large number, $\lim_{\tau\to\infty} P_\tau = P$. To do so, we make use of an exploration strategy, namely `probe`. To better understand the probing exploration policy, let $U(x) = \{1,\dots,|U(x)|\}$ be the set of available actions in state $x \in X$, and $\sigma(x)$ be the number of actions to be experienced in state $x \in X$. Before a new estimation phase starts, we set $\sigma(x) = |U(x)|$ for every state $x \in X$. At each time step of the model estimation phase, if $\sigma(x) > 0$, the centralized coordinator executes action $\sigma(x)$ in state x and decrements $\sigma(x)$; otherwise, he or she selects the action which minimizes the difference between the estimated and the optimized exploration policies (updated at the improvement phase), denoted $\hat{\pi}_\tau$ and π_τ, respectively: for any arbitrary $x \in X$ and vector of counts σ,

$$\texttt{probe}(x,\sigma) \doteq \begin{cases} \sigma(x), \text{ if } \sigma(x) > 0 \\ \arg\min_{u\in U(x)}\{\hat{\pi}_\tau(u|x) - \pi_\tau(u|x)\}, \text{ otherwise.} \end{cases}$$

If the state space forms a single positive recurrent class under any stationary policy, `probe` ensures every state-action pair gets visited at least once at each model estimation phase, in which case the estimation phase terminates. Otherwise, we shall impose a training budget $\tau_{\max}$ during each model estimation phase. Once the budget is exhausted the estimation phase stops—in that case, there is no guarantee of visiting every state-action pair. Next, the algorithm proceeds to the policy-improvement phase.

Each policy-improvement phase starts by computing an extended occupancy measure ζ_τ for the current estimate dynamics P_τ using our MILP formulations. Then, it calculates the state-dependent exploration policy as follows:

$$\pi_\tau(u|x) = \textstyle\sum_o \zeta_\tau(x,o,u) / \sum_{x,o} \zeta_\tau(x,o,u). \qquad (13)$$

Next, it ensures $\hat{\pi}_\tau \doteq \{\hat{\pi}_\tau(u|x)\}$ is a consistent estimator of π_τ, where $\hat{\pi}_\tau(u|x) \doteq w_\tau(x,u)/w_\tau(x)$, if $w_\tau(x) > 0$; and 0 otherwise. To this end, it explores the state space by selecting the action which minimizes the difference between the estimated and the optimized exploration policies, until $\|\hat{\pi}_\tau - \pi_\tau\|_\infty$ goes below a certain threshold. It is worth noticing that that this algorithm requires no hyper-parameter tuning. We present the pseudocode of our `probing` algorithm in the supplementary material.

7 Experiments

This section empirically demonstrates and validates the scalability of the proposed planning and learning approach w.r.t. the number of agents with $\alpha = 0.9$. We show that our planning and learning approach applies to n-agent Dec-POMDPs where no other MILP formulation does. We run our experiments on Intel(R) Xeon(R) CPU E5-2623 v3 3.00GHz.

7.1 ND-POMDPs

We conduct experiments on well-established benchmarks for evaluating n-agent Dec-POMDPs, *i.e.* network-distributed domains based on the sensor network applications Nair et al. [2005], which range from four to fifteen agents. The reader interested in the description of the benchmarks can refer to `http://teamcore.usc.edu/projects/dpomdp/`. To the best of our knowledge, no other MILP formulation can solve these domains, *e.g.,* Kumar et al. [2016] is inapplicable. Alternative approaches include: an extension of FB-HSVI for network-distributed domains Dibangoye et al. [2014a], unfortunately the only available formulation is dedicated for finite-horizon settings; and local search methods, *e.g.,* Kumar et al. [2011], which (i) can only provide local optima; (ii) trade theoretical guarantees for scalability w.r.t. the number of states and (iii) go beyond the scope of this paper. Instead, we target scalability w.r.t. the number of agents while preserving theoretical guarantees over a selected class of decentralized and deterministic ℓ-order Markov policies. Table 1 reports results on all tested n-agent domains. To validate the potential of this learning approach, we demonstrate one can learn the exact model and a corresponding policy in Figure 1.

Algorithm	$\lvert a\rvert$	Time	$J_\alpha(a;\nu)$
4 domain — $\lvert X\rvert = 12$, $n = 4$, $\lvert Z_i\rvert = 2$, *and* $2 \leq \lvert U_i\rvert \leq 3$			
$\mathcal{P}_4(\ell = 1)$	3×3	0.37s	752.492
$\mathcal{P}_4(\ell = 2)$	5×5	0.39s	752.492
4-star domain — $\lvert X\rvert = 12$, $n = 4$, $\lvert Z_i\rvert = 2$, *and* $2 \leq \lvert U_i\rvert \leq 3$			
$\mathcal{P}_4(\ell = 1)$	3×3	0.279s	568.642
$\mathcal{P}_4(\ell = 2)$	5×5	0.298s	568.642
5-star domain — $\lvert X\rvert = 12$, $n = 5$, $\lvert Z_i\rvert = 2$, *and* $2 \leq \lvert U_i\rvert \leq 3$			
$\mathcal{P}_4(\ell = 1)$	3×3	0.932s	407.821
$\mathcal{P}_4(\ell = 2)$	5×5	1.241s	407.821
5-P domain — $\lvert X\rvert = 12$, $n = 5$, $\lvert Z_i\rvert = 2$, *and* $2 \leq \lvert U_i\rvert \leq 3$			
$\mathcal{P}_4(\ell = 1)$	3×3	7.12s	334.536
$\mathcal{P}_4(\ell = 2)$	5×5	10.01s	334.536
7-H domain — $\lvert X\rvert = 12$; $n = 7$, $\lvert Z_i\rvert = 2$, *and* $2 \leq \lvert U_i\rvert \leq 3$			
$\mathcal{P}_4(\ell = 1)$	3×3	5.10s	199.773
$\mathcal{P}_4(\ell = 2)$	5×5	7.63s	199.773
15-3D domain — $\lvert X\rvert = 60$; $n = 15$, $\lvert Z_i\rvert = 2$, *and* $2 \leq \lvert U_i\rvert \leq 4$			
$\mathcal{P}_4(\ell = 1)$	3×3	1328.48s	409.069
$\mathcal{P}_4(\ell = 2)$	5×5	3002.19s	409.069
15-Mod domain — $\lvert X\rvert = 16$; $n = 15$, $\lvert Z_i\rvert = 2$, *and* $2 \leq \lvert U_i\rvert \leq 4$			
$\mathcal{P}_4(\ell = 1)$	3×3	27.1845s	571.642
$\mathcal{P}_4(\ell = 2)$	5×5	59.238s	571.642

Table 1: MILP results for n-agent ∞-horizon ND-POMDPs. Higher $J_\alpha(a;\nu)$ is better.

To summarize, our experiments illustrate the ability for our MILP formulations to quickly find finite-memory decentralized policies, that may serve as good approximations of the optimal decentralized policy. They also demonstrate the scalability with respect to the number of agents. Our MILP formulations optimally solve all network-distributed domains with up to fifteen agents. The main limitation is the lack of scalability w.r.t. the number of states, actions, observations, and hence histories. We argue that for domains where the double-exponential number of histories

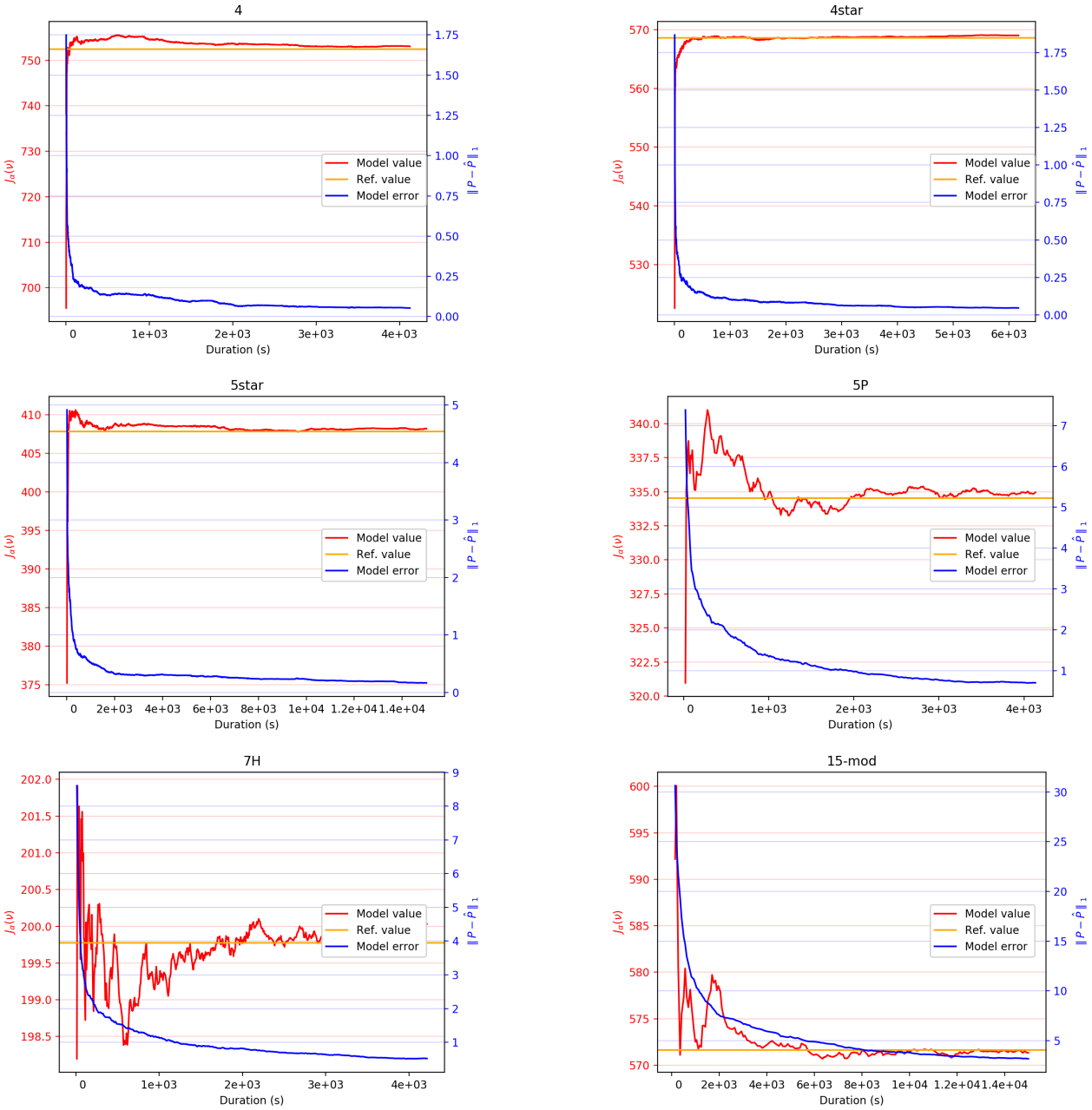


Figure 1: `Probing` results for n-agent ∞-horizon ND-POMDPs with $\alpha = 0.9$.

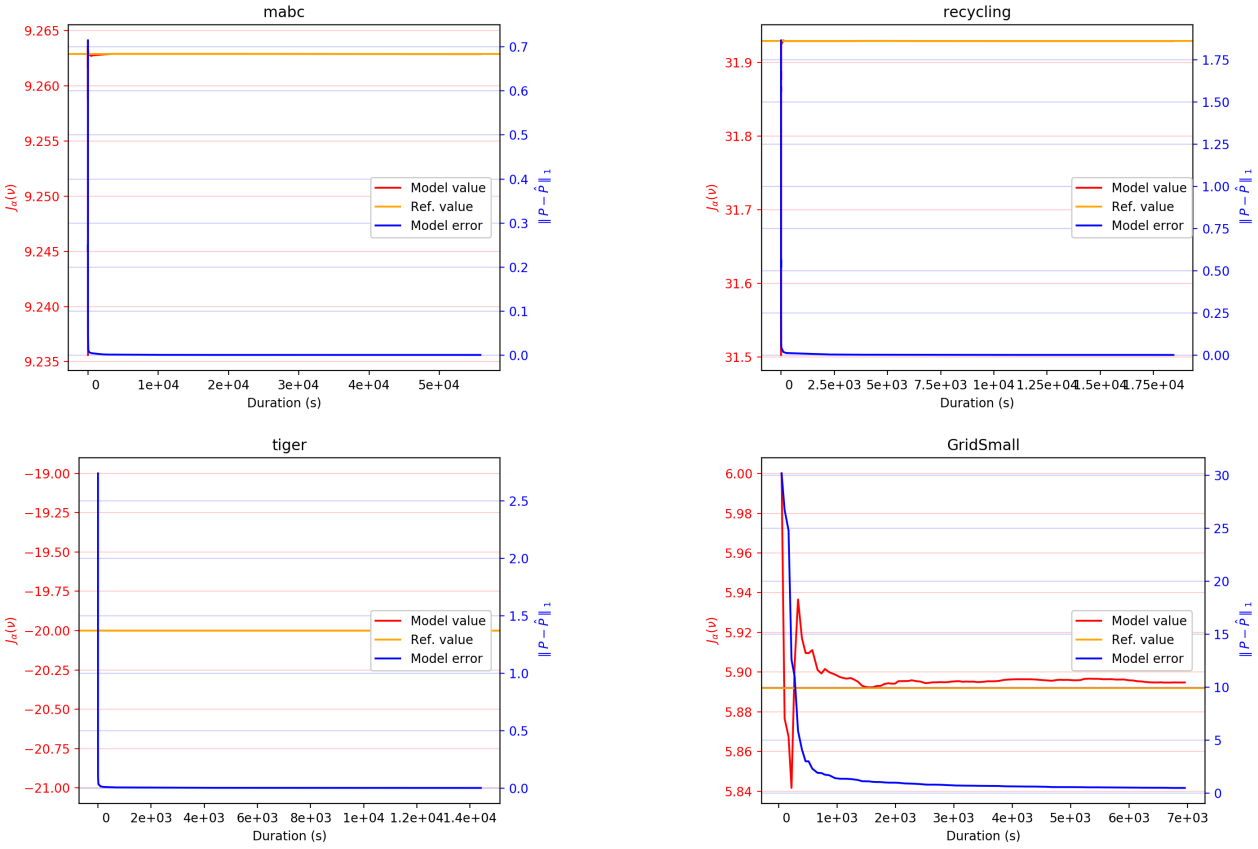


Figure 2: `Probing` results for n-agent ∞-horizon 2-agent Dec-POMDPs.

affect the scalability far more strongly than the number of states and actions, our approach is particularly useful. Otherwise, we plan to investigate learning a low-dimensional representation of the model of world making it possible to apply our approach. As for the learning part, we show our ability to quickly learn to find the target decentralized policy. None surprisingly, the process is slower in domains with more agents, states, actions and observations. We delay the problem of learning a model reduction for future work.

7.2 Two-agent Dec-POMDPs

Algorithm	$\|a\|$	Time	$J_\alpha(a;\nu)$
Broadcast ($\|X\|=4,\ \|U_i\|=2,\ \|Z_i\|=2$)			
$\mathcal{P}_2(\ell=1)$	3×3	0.01s	9.19
$\mathcal{P}_2(\ell=2)$	5×5	0.01s	9.2629
FB-HSVI	102	19.8s	9.271
FB-HSVI($\delta=0.01$)	435	7.8s	9.269
Kumar et al. [2016]	3×3	0.05s	9.1
Dec-tiger ($\|X\|=2,\ \|U_i\|=3,\ \|Z_i\|=2$)			
FB-HSVI($\delta=0.01$)	52	6s	13.448
FB-HSVI	25	157.3s	13.448
Kumar et al. [2016]	7×7	4.2s	13.4
MPBVI	231	<18000s	13.448
EM	6	142s	-16.3
$\mathcal{P}_2(\ell=2)$	5×5	0.01s	-20
Recycling robots ($\|X\|=4,\ \|U_i\|=3,\ \|Z_i\|=2$)			
$\mathcal{P}_3(\ell=1)$	3×3	0.01s	31.9291
FB-HSVI	109	2.6s	31.929
FB-HSVI($\delta=0.01$)	108	0s	31.928
Kumar et al. [2016]	3×3	1.1s	31.9
EM	2	13s	31.50
Meeting in a 3x3 grid ($\|X\|=81,\ \|U_i\|=5,\ \|Z_i\|=9$)			
$\mathcal{P}_3(\ell=1)$	10×10	0.19s	5.81987
FB-HSVI	108	67s	5.802
FB-HSVI($\delta=0.01$)	88	45s	5.794
Kumar et al. [2016]	10×10	4.4s	5.8
Box-pushing ($\|X\|=100,\ \|U_i\|=4,\ \|Z_i\|=5$)			
FB-HSVI($\delta=0.01$)	331	1715.1s	224.43
FB-HSVI($\delta=0.05$)	288	1405.7s	224.26
Kumar et al. [2016]	7×8	6.2s	181.2
$\mathcal{P}_2(\ell=1)$	6×6	0.06s	181.985
$\mathcal{P}_2(\ell=2)$	26×26	1.86s	197.607
Mars rover ($\|X\|=256,\ \|U_i\|=6,\ \|Z_i\|=8$)			
FB-HSVI($\delta=0.01$)	136	74.31s	26.94
FB-HSVI($\delta=0.2$)	149	85.72s	26.92
Kumar et al. [2016]	9×9	20.2s	23.8
$\mathcal{P}_2(\ell=1)$	9×9	2.48s	23.8302
EM	3	5096s	17.75

Table 2: Results for infinite-horizon domains with $\alpha = 0.9$. Higher $J_\alpha(a;\nu)$ is better. δ and ℓ denote the regret in Bellman's backup and the class of policies, respectively.

For the sake of completeness we also present our performances for 2-agent Dec-POMDPs, where we provide competitive results w.r.t. state-of-the-art methods. Experiments on 2-agent Dec-POMDPs were conducted on standard benchmarks as well as, all available at `masplan.org`. We use two of our 2-agent MILP formulations, *i.e.,* $\mathcal{P}_2(\ell)$ and $\mathcal{P}_3(\ell=1)$, for $\ell\in\{1,2\}$. Though our MILP formulations are guaranteed to find an optimal solution in the target class of policies, we do not expect them to do always better than the state-of-the-art solver FB-HSVI, since the latter achieves provably near-optimal performance on these benchmarks. Instead, these domains serve for the sanity check, assessing the quality of our solutions w.r.t. near-optimal ones. We also report performances from other decentralized POMDP solvers—*e.g.,* Kumar et al. [2016] and EM—though FB-HSVI demonstrated better performances on all tested 2-agent domains Dibangoye et al. [2014b].

Results for Kumar et al. [2016], EM were likely computed on different platforms, and, therefore, time comparisons may be approximate at best. Results for 2-agent domains can be seen in Table 2. In many tested 2-agent domains, low-order Markov policies achieve good performances. Hence our MILP formulations are competitive to state-of-the-art algorithms. In decentralized MDPs, *e.g.,* recycling robots and meeting in a 3x3 grid, decentralized and deterministic Markov policies are optimal, non-surprisingly $\mathcal{P}_3$ can find optimal solutions. However, in domains requiring more memory, *i.e.,* higher-order Markov policies, our MILP formulations may achieve poor performances, see for example Dec-tiger. The difficulty comes from the exponentially many joint histories to consider as ℓ increases. Overall, MILP formulations provide a simple yet efficient alternative to solving 2-agent domains, especially when (i) finite-memory policies can achieve good performances; and (ii) states, actions and observations are small. Finally, we successfully run the adaptive decentralized control approach on two small domains, *i.e.* recycling robots and broadcast see Figure 2, providing a strong theoretical support for this promising approach.

8 Conclusion

In this paper, we investigated MILP formulations, which proved useful for partially observable domains, but existing applications restrict to benchmarks with one or two agents. To overcome this limitation, we introduced a novel linearization property that allows us to reformulate nonlinear constraints from general decentralized partially observable Markov decision processes into linear ones. We further presented MILP formulations for general and special cases, including network-distributed and transition-independent problems. Our experiments on both standard 2-agent benchmarks as well as domains with a large number of agents illustrate the ability for our planning and learning approaches based on MILP formulations to find good approximate solutions often and sometimes optimal ones. Yet, the scalability, w.r.t. states, actions and observations, remains a major limitation. In future work, we shall generate an approximation of the extended occupancy measures within a parameterized class of functions to deal with the intractability of the exact MILP formulation, in a spirit similar to that of statistical regression. We shall draw inspiration from the literature of deep generative models, and more specifically (discrete) variational autoencoders Kingma and Welling [2014], Ha and Schmidhuber [2018].

References

E. Altman. *Constrained Markov Decision Processes*. Stochastic Modeling Series. Taylor & Francis, 1999. ISBN 9780849303821.

C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *JAAMAS*, 21(3):293–320, 2010.

R. Aras and A. Dutech. An Investigation into Mathematical Programming for Finite Horizon Decentralized POMDPs. *JAIR*, 37:329–396, 2010.

R. Becker, S. Zilberstein, V. R. Lesser, and C. V. Goldman. Solving Transition Independent Decentralized Markov Decision Processes. *JAIR*, 22:423–455, 2004.

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), 2002.

T. Berthold, S. Heinz, and M. E. Pfetsch. Nonlinear pseudo-boolean optimization: relaxation or propagation? Technical Report 09-11, ZIB, Takustr. 7, 14195 Berlin, 2009.

J. S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet. Optimally Solving Dec-POMDPs As Continuous-state MDPs. In *IJCAI*, pages 90–96, 2013.

J. S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet. Exploiting Separability in Multi-Agent Planning with Continuous-State MDPs. In *AAMAS*, pages 1281–1288, 2014a.

J. S. Dibangoye, O. Buffet, and F. Charpillet. Error-Bounded Approximations for Infinite-Horizon Discounted Decentralized POMDPs. In *ECML*, pages 338–353, 2014b.

C. V. Goldman and S. Zilberstein. Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis. *JAIR*, 22(1):143–174, 2004. ISSN 1076-9757.

D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, pages 2450–2462, 2018.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *stat*, 1050:1, 2014.

A. Kumar, S. Zilberstein, and M. Toussaint. Scalable Multiagent Planning Using Probabilistic Inference. In *AAAI*, pages 2140–2146, 2011.

A. Kumar, S. Zilberstein, and M. Toussaint. Probabilistic Inference Techniques for Scalable Multiagent Decision Making. *JAIR*, 53:223–270, 2015.

A. Kumar, H. Mostafa, and S. Zilberstein. Dual formulations for optimizing Dec-POMDP controllers. In *ICAPS*, pages 202–210, 2016.

H. Mostafa and V. Lesser. Compact mathematical programs for dec-mdps with structured agent interactions. In *UAI*, pages 523–530, 2011.

R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs. In *AAAI*, pages 133–139, 2005.

F. A. Oliehoek. Sufficient Plan-Time Statistics for Decentralized POMDPs. In *IJCAI*, 2013.

F. A. Oliehoek, M. T. J. Spaan, and N. A. Vlassis. Optimal and Approximate Q-value Functions for Decentralized POMDPs. *JAIR*, 32:289–353, 2008.

S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *JAAMAS*, 17(2):190–250, 2008.

D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *UAI*, 2005.

S. Witwicki and E. Durfee. Commitment-driven distributed joint policy search. In *AAMAS*, pages 75:1–75:8, 2007.

J. Wu and E. H. Durfee. Mixed-integer linear programming for transition-independent decentralized MDPs. In *AAMAS*, pages 1058–1060, 2006.

Contingent Planning Using Counter-examples From a Conformant Planner

Sébastien Piedade[1] Alban Grastien[2] Charles Lesire[1] Guillaume Infantes[3]

[1] ONERA, Toulouse, France
[2] Australian National University, Canberra, Australia
[3] JOLIBRAIN, Toulouse, France

sebastien.piedade@onera.fr

Résumé

Afin d'accomplir efficacement une mission de robots autonomes, le processus de prise de décision des robots doit être capable de traiter les incertitudes. Dans ce papier, on propose une approche novatrice de planification contingente, c'est à dire de planification sous incertitude à propos de l'état initial, qui inclut des tâches permettant d'obtenir des observations (partielles) sur l'état du monde. Cette approche contingente utilise un planificateur conformant, c'est à dire un planificateur qui n'est pas autorisé à faire des observations. Le principe général de notre approche est d'utiliser des contre-exemples retournés par le planificateur conformant quand aucun plan conformant n'est possible pour inclure des actions d'observations dans le plan, et réutiliser le planificateur conformant sur des sous-problèmes avec moins d'incertitude. On évalue cette approche en comparant ses résultats avec Contingent-FF, un planificateur contingent bien connu sur un ensemble de problèmes.

Mots Clef

planification contingente, décision, incertitude.

Abstract

Decision-making for autonomous robots in real world applications has to manage uncertainties in order to be able to efficiently accomplish a mission. In this paper, we propose a novel approach to contingent planning, i.e. to planning under uncertainty about the initial state, which includes tasks to get some (partial) observations about the world state. This contingent approach uses an underlying conformant planner, i.e. a planner that is not allowed to make observations. The general principle of our approach is to use counter-examples returned by the conformant planner when no conformant plan is available to include observation actions into the plan, and reuse the conformant planner on subproblems with less uncertainty. We evaluate this approach by comparing its results with respect to Contingent-FF, a well known contingent planner, on a set of benchmarks.

Keywords

Contingent planning, decision, uncertainty.

1 Introduction

In our world, where disasters are more and more frequent, fast and effective victims rescue has become a major issue. While robots are already used by first responders in such situations to access difficult terrains or hazardous areas, the next step would be to use *autonomous* robots, that would be able to adapt to the situation in order to provide fast and efficient response. This need for autonomous robotics in search and rescue has been emphasized by [6] and [23].
This capability to adapt to the environment requires us to embed into the robot platform some *decision-making process* able to reason about *uncertain states* of the environment, as stated by [19] while reporting about the use of robots in earthquake responses.
In this paper, we propose an algorithm for planning under uncertainty that settles in the *contingent planning* paradigm : uncertainty is represented by sets of possible states, and the objective is to find a conditional plan, i.e. a graph of actions containing branches allowing an online decision making influenced by the results of *observations* of some unknown parts of the environment. The originality of our approach is that we use in background a *conformant planner*, that looks for a plan without observation, that would allow to reach an objective from a set of possible initial states. We use this conformant planner iteratively, by asking it to solve subproblems, and we use counter-examples returned by this planner to insert observations in-between the conformant plans obtained for such subproblems. In this version of the approach, we try to perform the observation as close as possible from the failing action of a failing plan, assuming the fact that an observation could potentially be more efficient close to the issue.
The paper is organized as follows : next section presents some related works on planning under uncertainty. Then we present some background and notation about the problem formalisation, before describing our algorithm. Finally, we compare our approach with respect to contingent-FF, and we present some results on some academic bench-

marks.

2 Related Works

Various planning methods exist to handle uncertainty. We can separate these methods in different types depending on how uncertainty is defined. Replanning is a method consisting in computing a first plan without handling uncertainties and replanning if an event occurs during the execution of the plan [21, 26]. FF-Replan [26] is one of these methods in which the probabilities of the problem are determinized and a plan is computed with a classical planning method. If an unexpected state occurs during the execution of the plan, then the planner replans in the same determinization of the problem.

When uncertainty can be represented as probabilities on state transitions or action non-deterministic effects, probabilistic planning is commonly used. Among these probabilistic methods we can cite Markov Decision Processes (MDPs ; [24]) that model the problem as a fully observable stochastic system. The solution for a MDP is an optimal policy mapping the best action to each state of the MDP. This optimal policy can be found by various methods, like dynamic programming [5] and some of its variants like Value iteration method or Policy iteration [22]. When the problem is only partially observable, Partially Observable MDPs (POMDPs ; [18]) are used instead of MDPs. POMDPs introduce a belief over the environment state which is updated by some observations. This belief is updated by a function over state transition probabilities and observation probabilities. Some solving methods for MDPs and POMDPs follow a forward search approach generally making some sampling from the initial belief, then computing only a partial policy [15, 25, 20]. [10] have also proposed an architectural approach to compute partial policy online in bounded time by making assumptions on the current belief.

When uncertainty is expressed as a set of possible initial states of the system, symbolic planning methods can be used. If the agent is not able to perform any observation, conformant planning can be used to solve the planning problem [8, 9, 4, 14]. The plan returned by a conformant planner is generally an action sequence. Conformant-FF [8] is a conformant planner that explores the belief space using heuristic functions based on a relaxation of the problem actions, by ignoring the delete lists of their effects. Conformant-FF then uses the FF planner to compute a relaxed plan for each search state. CPCES [14] is another conformant planner that computes a conformant plan using only a subset of the initial states, which allows CPCES to reduce the problem to classical planning.

If the planning problem is (partially) observable, contingent planning can be used to solve the problem. Contingent planning [17, 2, 7] consists in computing a conditional plan containing branches allowing an online decision making influenced by the results of some observations of the system. The conditional plan returned by a contingent planner is generally represented as a decision tree. Contingent-FF [17] is a contingent planner that uses the same belief space representation as Conformant-FF. In Contingent-FF, the search space is an And-Or tree and the returned plan is a sub-tree where all leaves are goal states. Belief states are represented through action-observation sequences.

The approach proposed in this paper is part of the contingent planning approaches. Indeed, in the kind of application like search and rescue missions by autonomous robots, designing a model of the complete problem by a (PO)MDP is hard, as most of the uncertainty distributions are either unknown, or with the occurence of rare events. Moreover, conformant plans do not always exist for these problems, as some observations *must* be done to decrease the uncertainty. However, the approach we propose can be seen as a contingent meta-planner that uses a conformant planner to compute conformant plans for subproblems when such a conformant plan exists.

3 Background

In this section, we introduce some background and notations that we use later to present our approach. In a first part, we introduce notations to describe the problem, including a description of states and operators. In a second part, we introduce some notations about belief representation, that we use in our approach to manage uncertainty. In these definitions, all actions are assumed to be deterministic and the uncertainty is assumed to lie in the initial situation only. Note that it has been proven that non-deterministic effects can be eliminated by introducing artificial initial uncertainty [3].

3.1 Problem definition

The following notations are adapted from [13, 1].

Definition 1 (Planning Problem). A planning problem $\mathbb{P}$ is defined by a tuple $(\mathcal{L}, \mathcal{O}, I, G)$ where :

- $\mathcal{L} = \{p_1, ..., p_n\}$ is a finite set of proposition symbols ; a state s is then represented by a set of propositions that hold, i.e. that are true, in s ; propositions that do not hold in s are assumed to be false ; we note $\mathcal{W} = 2^{\mathcal{L}}$ the set of all possible world states ;
- $\mathcal{O}$ is a finite set of *operators*, partitioned into the set of *actions* A and the set of *observations* O ; each operator $op \in \mathcal{O}$ is defined by a precondition $pre(op) \subseteq \mathcal{L}$ and a set of effects *eff(op)* ;
- $I \subseteq \mathcal{W}$ is the set of possible initial states ;
- $G \subseteq \mathcal{L}$ is the set of propositions defining the goal.

Definition 2 (Action application). An action $a \in A$ is *applicable* in state $s \in \mathcal{W}$ if and only if its preconditions hold in s, i.e.

$$pre(a) \subseteq s \tag{1}$$

Each effect e of a (i.e., $e \in$ *eff(a)*) is defined by a triple $con(e) \subseteq \mathcal{L}, add(e) \subseteq \mathcal{L}, del(e) \subseteq \mathcal{L}$, where :

- $con(e)$ are the conditions in which e is applied (unconditional effects are defined by $con(e) = \emptyset$) ;

— $add(e)$ are the propositions that will be added to the state after applying e;
— $del(e)$ are the propositions that will be deleted from the state after applying e.

If a is applicable in s, then $T(s,a)$ is the transition function such that :

$$T(s,a) = s - \bigcup_{e \in \textit{eff(a)} \text{ s.t. } con(e) \subseteq s} del(e) \cup \bigcup_{e \in \textit{eff(a)} \text{ s.t. } con(e) \subseteq s} add(e). \quad (2)$$

We assume that the problem actions are not self-contradictory, i.e, when applying action a in state s, for two effects $e, e' \in \textit{eff(a)}$, if $p \in add(e)$ and $p \in del(e')$, then $con(e)$ and $con(e')$ are not both satisfied in state s. We have $con(e) \cup con(e') \not\subseteq s$. Consequently, we also assume that for each effect e, $add(e) \cap del(e) = \emptyset$.

Definition 3 (Observation application). An observation $o \in O$ is applicable in state $s \in \mathcal{W}$ if and only if its preconditions hold in s (see Eq. (1)). The effects o are defined by a proposition ($\textit{eff(o)} \in \mathcal{L}$) that is observed when applying o, i.e. whose truth value is known after applying o. The application of an observation o in state s has no effect over the state s, i.e.

$$T(s,o) = s \quad (3)$$

The application of an observation o has no effect on the world state, but it will have an effect on the belief the agent has on the current state. Note that we assume that an observation o observes only one proposition at a time without loss of generality.

3.2 Belief reasonning

As classically done in planning under uncertainty, we model the uncertain knowledge about the current state as a *belief* represented by a set of all the possible states (consistent with the actions/observations done so far). Following notations are taken or adapted from [1, 12].

Definition 4 (Belief State). The current belief $\mathcal{B} = \{s_1, \ldots, s_n\} \subseteq \mathcal{W}$ is the set of possible current states s_1, ..., s_n. The initial belief $\mathcal{B}_0$ corresponds to the possible initial states I of the problem.

Definition 5 (Action application on a belief). An action $a \in A$ is applicable in belief $\mathcal{B}$ if and only if a is applicable for each possible state in $\mathcal{B}$, i.e., iff :

$$\forall s \in \mathcal{B}, pre(a) \subseteq s \quad (4)$$

The effect of applying action a in $\mathcal{B}$ then results in a belief $\mathcal{B}'$ such that :

$$\mathcal{B}' = \{T(s,a), \text{ s.t. } s \in \mathcal{B}\} \quad (5)$$

where $T(s,a)$ is computed according to Eq. (2). By extension, the effect of applying a in $\mathcal{B}$ can be noted as $T(\mathcal{B}, a)$.

Definition 6 (Observation application on a belief). An observation $o \in O$ is applicable in belief $\mathcal{B}$ if and only if o is applicable for each possible state in $\mathcal{B}$ (see Eq. (4)). Let $\nu(o)$ be the observation result, i.e. the observed truth value of the effects of o. We note $\nu^+(o) \subseteq \mathcal{L}$ the set of observed proposition that hold in the current state, and $\nu^-(o) \subseteq \mathcal{L}$ the set of observed proposition that do not hold. The application of observation o in belief $\mathcal{B}$ do not modify the state itself, as described in Def. 3, but results in a new belief $T(\mathcal{B}, o)$ such that :

$$T(\mathcal{B}, o) = \{s, \text{ s.t. } s \in \mathcal{B} \wedge \nu^+(o) \subseteq s \wedge \nu^-(o) \cap s = \emptyset\} \quad (6)$$

As we assume that an observation observes only one proposition p (see Def. 3), either $\nu^+(o)$ or $\nu^-(o)$ is empty, the other being equal to p. Also note that Eq. (6) would work for observations whose effect has multiple propositions.

3.3 Conditional Plan

A conditional plan can be represented as a graph of operators, leading an initial belief to a resulting belief (then having a flow network structure). Branchings in this graph correspond to results of observations, depending whether the observed properties holds or not in the current belief.

Definition 7 (Conditional Plan). Given a problem $\mathbb{P} = (\mathcal{L}, \mathcal{O}, I, G)$, a conditional plan is inductively defined by the following facts :

— The empty plan ε is a conditional plan;
— (op) is a conditional plan $\forall op \in \mathcal{O}$;
— if π_1 and π_2 are conditional plans, then $\pi_1;\pi_2$ is a conditional plan representing the sequence of π_1 and π_2;
— if π_1, π_2 are conditional plans and $o \in O$ is an observation, then the plan $\textbf{if } o \textbf{ then } \pi_1 \textbf{ else } \pi_2$ is a conditional plan representing that according to the result of observation o, π_1 is executed if the observed proposition is true, otherwise π_2 is executed.

A conditional plan π is *executable* in a belief $\mathcal{B}$ if its root operator is applicable in $\mathcal{B}$, and if all operators in π are applicable in the belief corresponding to the result of their previous operators. We note $T(\mathcal{B}, \pi)$ the result of applying an executable conditional plan in belief $\mathcal{B}$. Using a similar inductive definition, we formally say that a conditional plan π is executable in a belief $\mathcal{B}$ if :

— $\pi = (op)$, $op \in \mathcal{O}$ such that op is applicable in $\mathcal{B}$ (see Eq. 4); then $T(\mathcal{B}, \pi) = T(\mathcal{B}, op)$;
— $\pi = \pi_1;\pi_2$, with π_1 applicable in $\mathcal{B}$ and π_2 applicable in $T(\mathcal{B}, \pi_1)$; then $T(\mathcal{B}, \pi) = T(T(\mathcal{B}, \pi_1), \pi_2)$;
— $\pi = \textbf{if } o \textbf{ then } \pi_1 \textbf{ else } \pi_2$, with o applicable in $\mathcal{B}$, π_1 is applicable in $\mathcal{B}^+$ and π_2 is applicable in $\mathcal{B}^-$, where $\mathcal{B}^+$ (resp. $\mathcal{B}^-$) $= T(\mathcal{B}, o)$ when $\nu^+(o)$ (resp. $\nu^-(o)$) $= eff(o)$; the result of applying π is then $T(\mathcal{B}, \pi) = T(\mathcal{B}^+, \pi_1) \bigcup T(\mathcal{B}^-, \pi_2)$.

A conditional plan π executable in I and that leads to G (i.e. $G \in T(I, \pi)$) is a *solution* to problem $\mathbb{P}$. We can notice that the definition of a plan in classical formalism is equivalent to the three first points of Def. 7.

4 Contingent Planning Algorithm

The proposed approach settles on the use of a conformant planner, that is asked to solve subproblems. To be used in our approach, the conformant planner must return either the conformant plan it has found, or, in case no conformant plan exists, a counter-example (i.e. an initial state that caused the failure) and an information about the reason of the failure on this counter-example, in the form of a plan (i.e. a sequence of actions) that fails for this counter-example.

Based on such a conformant planner, the principle of our approach is to give a problem to solve to the conformant planner, and in case of failure, use the counter-example and the failing plan to determine which observation to perform, and when, and then split the problem into subproblems taking this observation into account to reduce the uncertainty on the subproblems, and then ask the conformant planner to solve these subproblems. This process is used iteratively on the subproblems if the conformant planner fails in finding a solution. In this version of the approach, we try to perform the observation as close as possible from the failing action of the failing plan returned by the conformant planner, assuming the fact that an observation could potentially be more efficient close to the plan issue.

As a conformant planner, we use CPCES [14], that fulfills our assumption : it provides a counter-example and a failing plan in case of failure. Note that we could use any conformant planner returning the same kind of information, and for conformant planners that would return a failing proposition instead of a failing plan, we could integrate it with slight modification of the algorithm, without reconsidering the approach.

In the next paragraph, we present the CPCES algorithm and the data it provides. Then we present and describe the main algorithm of our approach, before analysing some properties.

4.1 CPCES

CPCES [14] is a conformant planner that follows an iterative approach, in which a deterministic planner, namely FF [16], is used to find a plan π (an action sequence) for a subset of the initial belief, and then the validity of this plan on the complete initial belief is checked by solving a SAT problem with Z3 [11]. If the plan is not valid, Z3 provides a counter-example γ, i.e. a possible initial state for which the plan is not valid. This counter-example is integrated to the initial subset, and FF is asked to solve it again. This process is used iteratively, starting from one single state of the belief, until either a valid plan is found, or FF finds no plan for the subset, in which case the counter-example and the previous plan found by FF are returned.

Algorithm 1 is the CPCES algorithm taken from [14] and adapted to the notations introduced in the previous section. FF is called to compute a new plan in line 9, Z3 is used to check the plan validity in line 4.

Algorithm 1 CPCES Algorithm

Input: $\mathbb{P} = (\mathcal{L}, A, I, G)$
Output: π, γ
1: $\mathcal{B} := \emptyset$
2: $\pi := \varepsilon$
3: **loop**
4: check validity of π
5: **if** π is a solution for $\mathbb{P}$ **then**
6: **return** $\pi, \emptyset$
7: let γ be a counter-example
8: $\mathcal{B} := \mathcal{B} \cup \{\gamma\}$
9: compute a new plan π' for $\mathbb{P}' = (\mathcal{L}, A, \mathcal{B}, G)$
10: **if** no such π' exists **then**
11: **return** π, γ
12: $\pi := \pi'$

4.2 Contingent planner

Algorithm 2 Contingent Planning Procedure

Input: $\mathbb{P} = (\mathcal{L}, \mathcal{O}, I, G)$
Output: π_c
1: $\pi, \gamma := \mathrm{conformantPlanner}(\mathbb{P})$
2: **if** $\gamma = \emptyset$ **then**
3: **return** π
4: $\mathcal{B}_o, o := \mathrm{findObservation}(I, \mathcal{O}, \pi, \gamma)$
5: $\pi_o := \mathrm{ContingentPlanning}((\mathcal{L}, \mathcal{O}, I, \mathcal{B}_o))$
6: $\mathcal{B}^+ := T(\mathcal{B}_o, o)$ *with* $\nu^+(o) =$ *eff(o)*
7: $\pi_p := \mathrm{ContingentPlanning}((\mathcal{L}, \mathcal{O}, \mathcal{B}^+, G))$
8: $\mathcal{B}^- := T(\mathcal{B}_o, o)$ *with* $\nu^-(o) =$ *eff(o)*
9: $\pi_n := \mathrm{ContingentPlanning}((\mathcal{L}, \mathcal{O}, \mathcal{B}^-, G))$
10: **return** $(\pi_o$;**if** o **then** π_p **else** $\pi_n)$

Algorithm 2 is our main contingent plan computation algorithm. This algorithm takes as input the contingent problem $\mathbb{P}$ and returns a contingent plan π_c. We first ask a conformant planner (in our implementation, CPCES as described in Alg. 1) to compute a conformant plan π (line 1). If such a conformant plan exists, (line 2), we return this plan. Otherwise, the conformant planner returns a counter-example γ and a plan π that fails for this counter-example. From these information, we look for an observation to include in the plan. The $\mathrm{findObservation}$ function (further detailed in Alg. 3) returns an observation o and a belief $\mathcal{B}_o$ in which this observation could be performed (line 4).

Finally, we recall our algorithm on the subproblems corresponding to : reaching the belief states in which to perform the observation from the initial state (line 5), and reaching the goal form both cases where the belief has been updated after a positive observation (line 7) and a negative observation (line 9). We finally return a plan made of the subplan

to reach $\mathcal{B}_o$ followed by a branching conditionned by the observation result (line 10).

Algorithm 3 findObservation

Input: $I, \mathcal{O}, \pi, \gamma$
Output: $(\mathcal{B}_o, o)$
1: $beliefList := [I]$
2: $\mathcal{B} := I$
3: **for** a in π **do**
4: **if** a applicable in γ **then**
5: $\gamma := T(\gamma, a)$
6: $\mathcal{B} := T(\mathcal{B}, a)$
7: $beliefList := beliefList + \mathcal{B}$
8: **else**
9: let $unsatPre$ be the unsatisfied preconditions of a
10: **break**
11: **for** p in $unsatPre$ **do**
12: let o be an observation for p in $\mathcal{O}$
13: **for** $\mathcal{B}_o$ in $beliefList$ **do**
14: **if** o applicable in $\mathcal{B}_o$ **then**
15: **return** $(\mathcal{B}_o, o)$
16: **return** $(\mathcal{B}, None)$

Algorithm 3 is used to determine which observation o we need to perform to discriminate the counter-example γ from the other possible states and in which belief $\mathcal{B}_o$ we need to perform the observation. Inputs are the set of operators $\mathcal{O}$, the failing plan π previously computed by the conformant plan and the counter-example state γ. Outputs are the observation o we need to perform and the belief $\mathcal{B}_o$ in which we need to perform the observation o.
We first look for the action a in π that is not applicable in γ (lines 4 to 7) by iteratively applying each action of the plan to the counter-example γ (line 5). We also keep track of the beliefs computed by the application of each action of π to the initial belief (lines 6 and 7). Once the failing action has been found we get the set of propositions $unsatPre$ in the preconditions of a that does not hold in the state γ (line 9). We can notice that $unsatPre$ will never be empty because there is necessarily a failing action in the failing plan π returned by the conformant planner. Each proposition of $unsatPre$ is a potential observable proposition allowing to discriminate the counter-example and the other states in which this proposition does not hold from the other possible states. We then try to find an observation able to observe one of these propositions in one of the belief computed in the belief list (lines 11 to 15). We scan the possible observations in $\mathcal{O}$ to find an observation performing an observation effect over the value of the proposition p (line 12).
Finally, we verify if the observation o is applicable in one of the computed beliefs in $beliefList$ (lines 13 to 15). If o is applicable in the current belief $\mathcal{B}_o$ then we return $\mathcal{B}_o$ and o. In the other case, we verify if the observation o is applicable in one of the previously computed beliefs in $beliefList$. If o is not applicable in any belief of $beliefList$ then we try to find another observation able to observe another proposition p of $unsatPre$. If there is no observation able to discriminate the counter-exemple, then we return $None$ meaning that there is no possible observation.

4.3 Theoretical evaluation

Our method is sound because if the algorithm returns a condition plan π to the problem, then π is a *solution* has defined in Sec. 3.3 (under the assumption that the underlying conformant planner is sound). If there is no solution to the problem then the method terminates without finding a plan.
The algorithm always terminates because the size of the search space decreases at each iteration due to the splitting of the search space after each observation.
However, our method is not complete, particularly because there is no backtracking. We are currently improving the algorithm with backtracking over the choice of observations.

5 Evaluation and Results

We have evaluated our algorithm by comparing its performance with respect to Contingent-FF [17] on a set of benchmarks provided by Contingent-FF. We had to limit the comparison to Contingent-FF only because other contingent planner do not accept PDDL problems as input. For this evaluation, we limited the computation time of the two approaches to 5 minutes, and we used the heuristic option of Contingent-FF that provided the fastest results (otherwise the solver times out on most of the benchmarks). Within these benchmarks, *(e)rovers* perfectly correpond to the kind of application mission we consider : a rover has to take samples or images of rocks and soil and communicate the data to a base in an environment where rocks and samples position and visibility are uncertain. Results are given on Tab. 1.
First, we can notice that for some benchmarks we find the same results as Contingent-FF, except for computation time, namely *ebtcs*, *grid/p2*, *egrid/p2*, *elogistics/p1,p3*. Sometimes neither Contingent-FF nor our approach are able to find a solution, like in *egrid/p3,p4*, where our approach does not succeed to find an applicable observation and Contingent-FF times out.
We can notice that Contingent-FF has clearly better results in *blocks* where our approach find plans with the same number of observations but with a bigger size and a longer depth. In *erovers/p4*, we obtain the same result in size as Contingent-FF, but our approach computes a longer plan in depth. Moreover in *erovers/p6*, Contingent-FF finds a solution with less observations, even if our solution is shorter.
We can observe that our approach is better than Contingent-FF in benchmarks where a conformant solution exists, namely *btcs*, *grid*, *rovers* and *logistics*. In that case, as we rely on CPCES, we find a conformant plan whereas Contingent-FF includes observations in its solu-

TABLE 1. Results of a comparison with Contingent-FF on some benchmarks. Computation times are given in seconds. TO indicates that the computation timed out after 5 min. NO indicates that the planner did not find an applicable observation. *Size* gives the number of actions in the plan, *depth* the maximal depth of the plan, and *observations* the number of observations. For our approach, we also compute the depth of the *shortest* path of the plan.

Problem	Contingent Planning with counter-examples					Contingent-FF			
	total	size	depth	shortest	observations	time	size	depth	observations
blocks/p3	0.94	6	4	3	1	0.00	6	4	1
blocks/p7	5.6	89	16	10	7	0.05	**55**	**9**	7
blocks/p11	6.4	169	29	20	7	0.43	**117**	**18**	7
blocks/p15	8.05	244	39	27	7	3.20	**163**	**25**	7
btcs/p10	0.76	19	19	19	**0**	0.02	19	10	9
btcs/p30	2.36	59	59	59	**0**	0.8	59	30	29
btcs/p50	8.13	99	99	99	**0**	9.79	99	50	49
btcs/p70	24.11	139	139	139	**0**	57.31	139	70	69
ebtcs/p10	6.21	19	10	2	9	0.01	19	10	9
ebtcs/p30	22.73	59	30	2	29	0.42	59	30	29
ebtcs/p50	56.8	99	50	2	49	4.93	99	50	49
ebtcs/p70	156.11	139	70	2	69	29.10	139	70	69
grid/p2	3.61	9	9	9	0	0.01	9	9	0
grid/p3	4.05	**19**	**19**	19	**0**	9.78	174	43	15
grid/p4	21.24	**45**	**45**	45	**0**	227	464	68	17
grid/p5	18.64	**31**	**31**	**31**	**0**	TO	-	-	-
egrid/p2	3.96	9	9	9	0	0.01	9	9	0
egrid/p3	NO	-	-	-	-	TO	-	-	-
egrid/p4	NO	-	-	-	-	TO	-	-	-
egrid/p5	73.24	**185**	**31**	**23**	**7**	TO	-	-	-
rovers/p2	0.38	**8**	**8**	8	**0**	0.00	13	10	1
rovers/p4	0.52	**13**	**13**	13	**0**	0.00	23	14	3
rovers/p6	0.86	**23**	**23**	23	**0**	0.11	448	66	11
rovers/p8	0.65	**23**	**23**	23	**0**	0.03	170	83	3
erovers/p2	1.09	**11**	**9**	5	1	0.00	13	10	1
erovers/p4	3.39	23	17	5	3	0.00	23	**14**	3
erovers/p6	15.52	**144**	**27**	21	11	0.09	346	48	**7**
erovers/p8	3.34	**44**	**21**	15	3	0.01	95	36	3
logistics/p1	0.39	**9**	9	9	**0**	0.01	10	7	1
logistics/p3	0.49	**14**	14	14	**0**	0.01	18	8	2
logistics/p5	0.58	**29**	29	29	**0**	0.054	172	26	7
logistics/p7	0.75	**31**	31	31	**0**	0.2	247	27	11
elogistics/p1	1.07	10	7	4	1	0.00	10	7	1
elogistics/p3	1.8	18	8	5	2	0.00	18	8	2
elogistics/p5	9.02	**138**	**22**	20	7	0.12	172	26	7
elogistics/p7	10.63	**185**	26	21	11	0.13	247	26	11

tion. Moreover, it generally results in finding a shorter plan, except for *logistics* and *btcs* where Contingent-FF finds a plan with a shorter depth. In problems like *elogistics/p5,p7* and *erovers/p2,p8*, we find plans having the same number of observations than Contingent-FF, but the plan we find are shorter in size and depth. Moreover our approach succeed in solving *egrid/p5* problem while Contingent-FF times out.

Finally, we generally get some good results, except regaring the computation time needed to solve some of the problems. First, we can notice these computation times have the same order of magnitude than Contingent-FF and do not seem to grow exponentially when increasing the size of the problems. Second, this computation time partly comes from the fact that we use CPCES as a "black-box" conformant planner, itself considering FF as a "black-box" planner. This induces a lot of access to files for writing/reading problems for this solvers during our process, while Contingent-FF does all the computation in memory.

As a conclusion, we can notice that the quality of plans

obtained for the *rovers* and most of the *erovers* benchmarks clearly outperform the results of Contingent-FF. We obtain similar results in *(e)logistics* that also include features of our autonomous robot mission.

6 Conclusion

In this paper, we have proposed a new contingent planner with an original approach, as we use a conformant planner to find conformant sub-plans when possible. Our approach consists in asking CPCES, a conformant planner, to solve subproblems, iteratively built by identifying the necessary observations to perform from previous CPCES counter-examples.

Such an approach is of high interest for decision-making in autonomous robot applications, where we have to cope with uncertainty in the environment (partially known terrains, unknown number and state of searched objects, ...). Such a contingent approach has the advantage to compute offline a conditional plan, then reducing the computation time needed by online replaning approaches, and moreover computing robust plans, as the plan would be "piecewise conformant", then locally robust to all contingencies, and observations are required only when necessary to reach the mission objective.

In order to evaluate our approach, we compared with Contingent-FF on a set of benchmarks. Despite higher computation time (that could be reduced by optimizing the algorithm implementation), we get some concluding results on the plan quality, especially on the benchmarks that model the kind of autonomous robots missions we want to solve. We generally find solutions either with less observations, or with less actions in the plan.

Future works consist first in improving the completeness of our method by performing a backtracking in the failing plan computation and in the observation computation process if we fail to find an observation applicable in a belief computed from the current failing plan. Second, we will apply our method to real-world autonomous robot mission consisting in exploring outdoor partially known environments and find possible intruders in order to protect a critical facility.

Références

[1] A. Albore and H. Geffner. Acting in partially observable environments when achievement of the goal cannot be guaranteed. In *Proc. of ICAPS Workshop on Planning and Plan Execution for Real-World Systems*. Citeseer, 2009.

[2] A. Albore, H. Palacios, and H. Geffner. A translation-based approach to contingent planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Providence, RI, USA, July 2009.

[3] A. Albore, H. Palacios, and H. Geffner. Compiling uncertainty away in non-deterministic conformant planning. In *European Conference on Artificial Intelligence (ECAI)*, Lisbon, Portugal, August 2010.

[4] A. Albore, M. Ramirez, and H. Geffner. Effective heuristics and belief tracking for planning with incomplete information. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Freiburg, Germany, June 2011.

[5] R. Bellman. Dynamic programming. *Science*, 153(3731) :34–37, 1966.

[6] A. Birk and S. Carpin. Rescue robotics — a crucial milestone on the road to autonomous systems. *Advanced Robotics*, 20(5) :595–605, 2006.

[7] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Thessaloniki, Greece, September 2009.

[8] R. Brafman and J. Hoffmann. Conformant planning via heuristic forward search : A new approach. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Whistler, Canada, June 2004.

[9] D. Bryce, S. Kambhampati, and D. E. Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research (JAIR)*, 26 :35–99, 2006.

[10] C. P. C. Chanel, A. Albore, J. T'Hooft, C. Lesire, and F. Teichteil-Königsbuch. Ample : an anytime planning and execution framework for dynamic and uncertain problems in robotics. *Autonomous Robots*, 43(1) :37–62, 2019.

[11] L. De Moura and N. Bjørner. Z3 : An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[12] P. E. U. de Souza, C. P. C. Chanel, and F. Dehais. Momdp-based target search mission taking into account the human operator's cognitive state. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 729–736. IEEE, 2015.

[13] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning : theory and practice*. Elsevier, 2004.

[14] A. Grastien and E. Scala. Intelligent belief state sampling for conformant planning. In *IJCAI*, pages 4317–4323, 2017.

[15] E. Hansen and S. Zilberstein. LAO* : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence Journal (AIJ)*, 129(1-2) :35–62, 2001.

[16] J. Hoffmann. Ff : The fast-forward planning system. *AI magazine*, 22(3) :57–57, 2001.

[17] J. Hoffmann and R. Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS*, volume 2005, 2005.

[18] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2) :99–134, 1998.

[19] H. Kitano and S. Tadokoro. Robocup rescue : A grand challenge for multiagent and intelligent systems. *AI magazine*, 22(1) :39–39, 2001.

[20] H. Kurniawati, D. Hsu, and W. S. Lee. SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Robotics : Science and Systems (RSS)*. Zurich, Switzerland, July 2008.

[21] U. Kuter, D. Nau, E. Reisner, and R. Goldman. Using classical planners to solve nondeterministic planning problems. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Sydney, Australia, September 2008.

[22] Mausam and A. Kolobov. Planning with markov decision processes : An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1) :1–210, 2012.

[23] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen. Search and rescue robotics. *Springer handbook of robotics*, pages 1151–1173, 2008.

[24] M. L. Puterman. *Markov decision processes : discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[25] F. Teichteil-Königsbuch, U. Kuter, and G. Infantes. Incremental plan aggregation for generating policies in MDPs. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, Toronto, Canada, May 2010.

[26] S. W. Yoon, A. Fern, and R. Givan. Ff-replan : A baseline for probabilistic planning. In *ICAPS*, volume 7, pages 352–359, 2007.

La logique dynamique des affectations propositionnelles parallèles DL-PPA et son application à la planification parallèle

Andreas Herzig[1] Frédéric Maris[2] Julien Vianey[2]

[1] IRIT, CNRS
[2] IRIT, Université Toulouse 3 - Paul Sabatier

{andreas.herzig, frederic.maris, julien.vianey}@irit.fr

Résumé

Nous introduisons une logique dynamique avec une composition parallèle et deux types de composition non déterministe, exclusive et inclusive. Nous montrons qu'à la fois la vérification de modèle et le problème de satisfiabilité sont PSPACE-complets et appliquons notre logique à la planification classique séquentielle et parallèle où les actions ont des effets conditionnels.

Mots Clef

logique dynamique, planification parallèle, effets conditionnels.

Abstract

We introduce a dynamic logic with parallel composition and two kinds of nondeterministic composition, exclusive and inclusive. We show PSPACE completeness of both the model checking and the satisfiability problem and apply our logic to sequential and parallel classical planning where actions have conditional effects.

Keywords

dynamic logic, parallel planning, conditional effects.

1 Introduction

De nombreux auteurs ont étudié la manière dont les problèmes de planification et leurs solutions peuvent être représentés en logique dynamique propositionnelle PDL, incluant notamment des travaux sur la planification conformante et contingente et la planification avec des actions épistémiques [Spalazzi and Traverso, 2000, Andersen et al., 2012, Li et al., 2017, Bolander et al., 2018, Cong et al., 2018]. Cependant et à notre connaissance, il n'a pas encore été étudié comment la planification avec des actions simultanées peut être capturée dans la logique dynamique. La raison en est probablement qu'il n'y a pas de consensus sur la sémantique des actions parallèles dans la communauté PDL : il y a des propositions pour interpréter la composition parallèle comme un entrelacement [Mayer and Stockmeyer, 1996, Benevides and Schechter, 2014], comme une intersection [Balbiani and Vakarelov, 2003], et en tant que relation entre états et ensembles d'états [Peleg, 1987, Goldblatt, 1992]. Il existe également des extensions avec des modalités issues des logiques de séparation des ressources [Balbiani and Boudou, 2018, Benevides et al., 2011, Veloso et al., 2014].

Nous prenons ici une route différente et nous nous basons sur une version simple de la logique dynamique où les programmes atomiques sont des affectations de formules à des variables propositionnelles. La logique dynamique des affectations propositionnelles DL-PA a de nombreuses applications en représentation des connaissances [Herzig, 2014], en particulier la planification classique [Herzig et al., 2014], et elle est considérablement plus simple que PDL. En particulier, la complexité est beaucoup plus faible : la vérification de la satisfiabilité est dans PSPACE. Nous ajoutons un opérateur de composition parallèle à DL-PA, ainsi qu'un opérateur de composition non déterministe inclusive (par opposition à la composition non déterministe exclusive de PDL).

Afin de résoudre des tâches de planification avec des plans parallèles, plusieurs notions d'interférence ont été proposées [Knoblock, 1994, Dimopoulos et al., 1997]. Nous choisissons le cadre des actions parallèles indépendantes introduites dans le planificateur GRAPHPLAN [Blum and Furst, 1997] où deux actions interfèrent si l'une supprime une précondition ou un effet de l'autre. Cette définition plutôt restrictive est utilisée dans la plupart des approches de la planification classique parallèle. Les actions sans interférence peuvent être exécutées dans n'importe quel ordre séquentiel avec exactement le même résultat.

Le document est organisé comme suit. Nous étendons DL-PA à DL-PPA dans la section 2 et montrons dans la section 3 que la complexité reste dans PSPACE. Dans la section 4, nous montrons comment la planification séquentielle et parallèle ainsi que leurs versions bornées peuvent être traduites polynomialement en DL-PPA. La section 5 conclut.

2 DL-PA et DL-PPA

Notre logique dynamique des affectations propositonnelles parallèles DL-PPA étend la logique dynamique des affectation propositionnelles DL-PA par deux opérateurs : un opérateur de composition parallèle $\sqcap$ et un nouvel opérateur de composition non déterministe $\sqcup$. La distinction entre $\sqcup$ et l'opérateur standard de composition non déterministe $\cup$ est similaire à celle entre les disjonctions inclusive et exclusive : l'interprétation de $\pi_1 \cup \pi_2$ est "faire soit π_1 soit π_2 ", tandis que celle de $\pi_1 \sqcup \pi_2$ est "faire soit π_1, soit π_2, emph soit les deux". Nous appelons l'ancien choix non déterministe *exclusif* et le dernier choix non déterministe *inclusif*. L'interprétation de la composition parallèle $\pi_1 \sqcap \pi_2$ est que les deux programmes sont exécutés sur des copies locales des variables, puis l'état résultant est obtenu en fusionnant ces états locaux : $\pi_1 \sqcap \pi_2$ échoue si deux variables affectées se voient attribuer des valeurs de vérité différentes par π_1 et π_2 ; sinon, les nouvelles valeurs de vérité l'emportent sur les anciennes.

2.1 Langage

Le langage de DL-PPA est construit à partir d'un ensemble infini dénombrable $\mathbb{P}$ de variables propositionnelles. Les programmes π et les formules φ sont définis par la grammaire

$$\begin{array}{lll} \varphi & ::= & p \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi \\ \pi & ::= & p{\leftarrow}\varphi \mid \varphi? \mid \pi;\pi \mid \pi \cup \pi \mid \pi \sqcup \pi \mid \pi \sqcap \pi \mid \pi^* \end{array}$$

où p appartient à $\mathbb{P}$. La formule $\langle\pi\rangle\varphi$ se lit "il y a une exécution possible de π telle que φ soit vraie par la suite". Le programme $p{\leftarrow}\varphi$ affecte la valeur de vérité de φ à p ; par exemple, $p{\leftarrow}\neg p$ change la valeur de vérité de p. $\varphi?$ vérifie que φ est vrai (ce qui échoue lorsque φ est faux). $\pi_1;\pi_2$ exécute séquentiellement π_1 et π_2. $\pi_1 \cup \pi_2$ choisit de façon non déterministe entre l'exécution de π_1 ou π_2 ; $\pi_1 \sqcup \pi_2$ choisit de manière non déterministe entre l'exécution de π_1, ou π_2, ou des deux. $\pi_1 \sqcap \pi_2$ est la composition parallèle de π_1 et π_2. L'ensemble de toutes les formules est noté $Fml_{\text{DL-PPA}}$. Le langage de DL-PA est le fragment de DL-PPA sans $\sqcup$ et $\sqcap$.

L'ensemble des variables propositionnelles apparaissant dans une formule φ est noté $\mathbb{P}_\varphi$; de même, l'ensemble de variables apparaissant dans un programme π est noté $\mathbb{P}_\pi$. Par exemple, $\mathbb{P}_{p\leftarrow q\vee r} = \{p, q, r\}$.

La *longueur* des formules de DL-PPA et des programmes de DL-PPA est le nombre de symboles requis pour les écrire, en excluant les parenthèses et en considérant que la longueur des variables propositionnelles est 1. Nous les notons $\ell(\varphi)$ et $\ell(\pi)$.

Nous utilisons les abréviations standard telles que $\top$ et $\varphi \to \psi$, plus des itérations de programmes au plus n fois, définies récursivement par $\pi^{\leq 0} = \top?$ et $\pi^{\leq n+1} = \top? \cup (\pi;\pi^{\leq n})$. Pour $P = \{p_1, \ldots, p_n\}$, $\bigsemicolon_{p_i\in P} p_i{\leftarrow}\varphi_i$ est un raccourci pour $p_1{\leftarrow}\varphi_1;\cdots;p_n{\leftarrow}\varphi_n$. Nous l'identifions avec $\top?$ si $P = \emptyset$. Nous devons être prudents ici car la composition séquentielle ';' n'est pas commutative ; par exemple, l'interprétation de $p{\leftarrow}q;q{\leftarrow}p$ sera différent de celui de $q{\leftarrow}p;p{\leftarrow}q$. Nous nous assurerons que l'ordre n'a pas d'importance.

2.2 Sémantique de DL-PPA

La sémantique est donnée en termes de valuations (états) qui sont des sous-ensembles de $\mathbb{P}$. Ainsi, l'ensemble de toutes les valuations est $2^{\mathbb{P}}$. Nous utilisons $V, V', W, U, \ldots$ pour désigner des valuations.

$$\begin{aligned} \|p\|_{\text{DL-PA}} &= \{V \ : \ p \in V\} \\ \|\langle\pi\rangle\varphi\|_{\text{DL-PA}} &= \{V \ : \ \text{il existe } V' \text{ tel que } (V,V') \in \|\pi\|_{\text{DL-PA}} \\ &\qquad\qquad \text{et } V' \in \|\varphi\|_{\text{DL-PA}}\} \\ \|p{\leftarrow}\varphi\|_{\text{DL-PA}} &= \{(V, V\cup\{p\}) \ : \ V \in \|\varphi\|_{\text{DL-PA}}\} \cup \\ &\qquad \{(V, V\setminus\{p\}) \ : \ V \notin \|\varphi\|_{\text{DL-PA}}\} \\ \|\varphi?\|_{\text{DL-PA}} &= \{(V,V) \ : \ V \in \|\varphi\|_{\text{DL-PA}}\} \\ \|\pi;\pi'\|_{\text{DL-PA}} &= \|\pi\|_{\text{DL-PA}} \circ \|\pi'\|_{\text{DL-PA}} \\ \|\pi \cup \pi'\|_{\text{DL-PA}} &= \|\pi\|_{\text{DL-PA}} \cup \|\pi'\|_{\text{DL-PA}} \\ \|\pi^*\|_{\text{DL-PA}} &= (\|\pi\|_{\text{DL-PA}})^* = \bigcup_{k\in\mathbb{N}_0} (\|\pi\|_{\text{DL-PA}})^k \end{aligned}$$

Les formules et les programmes sont interprétés par récurrence mutuelle. Dans DL-PA, l'interprétation d'une formule φ était un ensemble de valuations $\|\varphi\|_{\text{DL-PA}} \subseteq 2^{\mathbb{P}}$ et l'interprétation d'un programme π une relation binaire sur les valuations $\|\pi\|_{\text{DL-PA}} \subseteq 2^{\mathbb{P}} \times 2^{\mathbb{P}}$; les principales clauses sont rappelées ci-dessus.

En revanche, dans DL-PPA, l'interprétation de π est une relation ternaire sur l'ensemble des valuations $\|\pi\| \subseteq 2^{\mathbb{P}} \times 2^{\mathbb{P}} \times 2^{\mathbb{P}}$. Lorsque $(V, U, W) \in \|\pi\|$ alors il y a une exécution de π de l'état V à l'état U en affectant les variables dans W. La définition est ici encore par récurrence mutuelle et les clauses principales sont données dans le tableau 1 (les autres sont standards).

Commentons les clauses qui sont nouvelles par rapport à DL-PA. La sémantique de l'affectation $p{\leftarrow}\varphi$ est : p est rendu vrai si φ est vrai et est rendu faux si φ est faux, et dans les deux cas, l'ensemble des variables affectées est le singleton $\{p\}$.

La sémantique de la composition parallèle est que chaque sous-programme π_k est exécuté localement ; Nous vérifions ensuite si les modifications (en termes de variables affectées) sont compatibles : c'est le cas lorsque toutes les variables affectées par les deux sous-programmes (à savoir les variables dans $W_1 \cap W_2$) ont été affectées avec la même valeur de vérité. Si ce n'est pas le cas, la composition parallèle échoue ; autrement la valeur résultante U est calculée en rassemblant (1) la partie inchangée de V (c'est-à-dire $V \setminus W$), (2) les mises à jour de π_1 (c'est-à-dire $U_1 \cap W_1$), (3) les mises à jour de π_2 (c'est-à-dire $U_2 \cap W_2$). De plus, l'ensemble des variables W affectées par la composition parallèle est l'union de celles affectées par les sous-programmes.

La sémantique de la composition non déterministe inclusive $\pi_1 \sqcup \pi_2$ est, comme annoncé, la composition non déterministe exclusive des trois programmes π_1, π_2 et $\pi_1 \sqcap \pi_2$.

$$
\begin{aligned}
\|p\| &= \{V \ : \ p \in V\} \\
\|\langle\pi\rangle\varphi\| &= \{V \ : \ \text{il existe } U, W \text{ tels que } (V,U,W) \in \|\pi\| \text{ et } U \in \|\varphi\|\} \\
\|p{\leftarrow}\varphi\| &= \{(V, V\cup\{p\}, \{p\}) \ : \ V \in \|\varphi\|\} \cup \{(V, V\backslash\{p\}, \{p\}) \ : \ V \notin \|\varphi\|\} \\
\|\varphi?\| &= \{(V,V,\emptyset) \ : \ V \in \|\varphi\|\} \\
\|\pi_1;\pi_2\| &= \{(V,U,W) \ : \ \text{il existe } U_1, W_1, W_2 \text{ tels que } (V,U_1,W_1) \in \|\pi_1\|, (U_1,U,W_2) \in \|\pi_2\| \text{ et } W = W_1 \cup W_2\} \\
\|\pi_1 \cup \pi_2\| &= \|\pi_1\| \cup \|\pi_2\| \\
\|\pi_1 \sqcup \pi_2\| &= \|\pi_1\| \cup \|\pi_2\| \cup \|\pi_1 \sqcap \pi_2\| \\
\|\pi_1 \sqcap \pi_2\| &= \left\{(V,U,W) \ : \ \begin{array}{l} \text{il existe } U_1, W_1, U_2, W_2 \text{ tels que } (V,U_1,W_1) \in \|\pi_1\|, (V,U_2,W_2) \in \|\pi_2\|, \\ W_1 \cap W_2 \cap U_1 = W_1 \cap W_2 \cap U_2, U = (V\backslash W) \cup (U_1 \cap W_1) \cup (U_2 \cap W_2), \text{ et } W = W_1 \cup W_2 \end{array} \right\} \\
\|\pi^*\| &= \bigcup_{k\in\mathbb{N}_0} \|\pi\|^k
\end{aligned}
$$

Table 1 – Interprétation des programmes DL-PPA

Voici quelques exemples d'interprétations de programmes :

$$
\begin{aligned}
\|p{\leftarrow}\bot\| &= \{(V, V\backslash\{p\}, \{p\}) \ : \ V \subseteq \mathbb{P}\} \\
\|\top? \sqcap p{\leftarrow}\bot\| &= \|p{\leftarrow}\bot\| \\
\|p{\leftarrow}\top \sqcap p{\leftarrow}\bot\| &= \emptyset \\
\|p{\leftarrow}\top \sqcap q{\leftarrow}\bot\| &= \{(V, (V\backslash\{q\})\cup\{p\}, \{p,q\}) \ : \ V \subseteq \mathbb{P}\} \\
\|p{\leftarrow}p \sqcap p{\leftarrow}\bot\| &= \{(V, V, \{p\}) \ : \ V \subseteq \mathbb{P} \text{ et } p \notin V\}
\end{aligned}
$$

Proposition 1. *Soit π un programme de* DL-PPA *et soit $(V,U,W) \in \|\pi\|$. Alors $W \subseteq \mathbb{P}_\pi$, et $V \subseteq \mathbb{P}_\pi$ implique $U \subseteq \mathbb{P}_\pi$.*

La proposition ci-dessus affirme que dans l'interprétation d'un programme π, la troisième composante W est un sous-ensemble de l'ensemble de variables se produisant dans π. Cela peut en fait être restreint un peu plus : W est en fait un sous-ensemble de l'ensemble de variables se trouvant sur le côté gauche des affectations de π.
Le résultat suivant indique que lors de l'interprétation de π, les variables n'apparaissant pas dans π n'ont pas d'importance en ce qui concerne les deux premières composantes V et U d'une interprétation.

Proposition 2. *Soit π un programme de* DL-PPA*. Soit P un ensemble de variables dont aucune n'apparaît dans π, c'est-à-dire telles que $P \cap \mathbb{P}_\pi = \emptyset$. Alors $(V\cup P, U\cup P, W) \in \|\pi\|$ si et seulement si $(V\backslash P, U\backslash P, W) \in \|\pi\|$.*

Une formule φ est *satisfiable* si et seulement si $\|\varphi\| \neq \emptyset$.
Grâce aux deux propositions ci-dessus, lors de la vérification de la validité ou de la satisfiabilité d'une formule φ, il suffit de vérifier les triplets (V,U,W) dont les éléments sont tous des sous-ensembles de $\mathbb{P}_\varphi$.
Lorsque les variables de π_1 et π_2 sont disjointes, ils peuvent être séquentialisés :

Proposition 3. *Soient π_1 et π_2 deux programmes tels que $\mathbb{P}_{\pi_1} \cap \mathbb{P}_{\pi_2} = \emptyset$. Alors $\|\pi_1 \sqcap \pi_2\| = \|\pi_1;\pi_2\| = \|\pi_2;\pi_1\|$.*

Tout comme dans DL-PA [Balbiani et al., 2014], l'étoile de Kleene peut être éliminée dans DL-PPA : intuitivement, lorsqu'il existe une exécution de π^* allant de x à y alors il est possible de passer de x à y en pas plus de $2^{|\mathbb{P}_\pi|}$ itérations de π. (En effet, s'il y a un parcours plus long, il passe nécessairement par la même valuation deux fois et il doit donc y avoir un raccourci.)

Proposition 4. *Pour tout programme π de* DL-PPA*, $\|\pi^*\| = \|\pi^{\leq 2^{|\mathbb{P}_\pi|}}\|$.*

Le résultat ci-dessus peut être renforcé : Nous pourrions remplacer $|\mathbb{P}_\pi|$ par le nombre de programmes atomiques apparaissant dans π ; encore mieux, nous pourrions le remplacer par le nombre de variables distinctes apparaissant à gauche des affectations de π. Par exemple, $(p{\leftarrow}q\vee r)^*$ peut alors être réduit à $(p{\leftarrow}q\vee r)^{\leq 2}$ au lieu de $(p{\leftarrow}q\vee r)^{\leq 8}$.

2.3 Compteurs

Dans DL-PA, Nous pouvons implémenter des compteurs de k bits (voir par exemple [Balbiani et al., 2013, Section II.C]), et ceci est transféré vers DL-PPA. L'encodage d'un nombre binaire de longueur k nécessite un vecteur de $\lceil \log k \rceil$ variables propositionnelles. Nous représentons l'état du compteur par une formule ct, qui représente la conjonction de ces $\lceil \log k \rceil$ variables ou leurs négations. De plus, nous utilisons les abréviations suivantes :

- $ct{\leftarrow}0$ est un programme qui réinitialise ct : toutes les variables du compteur sont définies à la valeur faux ;
- Pour chaque entier i, $ct{=}i$ est une formule qui est vraie si et seulement si la valeur encodée par ct est i ;
- $ct{+}{+}$ incrémente la valeur de ct.

Nous supposons que les variables utilisées dans un compteur ne sont pas utilisées ailleurs, afin qu'elles n'interfèrent pas avec d'autres programmes.
Les compteurs nous permettent de formuler de manière plus compacte notre abréviation $\pi^{\leq k}$ dont l'expansion a une longueur exponentielle en la taille de la représentation de k : en utilisant un compteur de $\lceil \log k \rceil$ bits, nous pouvons identifier $\pi^{\leq k}$ avec

$$ct{\leftarrow}0; (\neg ct{=}k\,?; \pi; ct{+}{+})^*$$

dont la longueur est linéaire en $\ell(\pi) + \log k$. En effet, bien que ces deux programmes n'aient pas la même interpréta-

tion, ils se comportent de la même façon en ce qui concerne les variables de π.

Proposition 5. *Pour les valuations $V, U, W \subseteq \mathbb{P}_\pi$, nous avons $(V, U, W) \in \|\pi^{\leq k}\|$ si et seulement si*

$(V, U \cup U', W \cup W') \in \|ct{\leftarrow}0; (\neg ct{=}k\,?; \pi; ct{+}{+})^*\|$

pour certains sous-ensembles U' et W' de l'ensemble des variables de compteur.

3 Réduction de DL-PPA à DL-PA

Nous allons maintenant donner une réduction polynomiale des formules et programmes de DL-PPA à DL-PA. Notre traduction élimine $\sqcap$ et $\sqcup$ grâce à l'introduction de (plusieurs types de) copies de variables propositionnelles.

Premièrement, la copie δ_p de p stocke que p a été assigné ; cela nous permettra de simuler la troisième composante W de l'interprétation des programmes qui conserve la trace des variables affectées.

Deuxièmement, lors de la traduction d'une composition parallèle nous séquentialisons l'exécution parallèle de deux programmes π_1 et π_2, et pour le faire en toute sécurité, nous laissons chacun travailler sur des copies locales des variables p, respectivement notées $(p)^1$ et $(p)^2$.

La traduction d'une composition non déterministe inclusive $\pi_1 \sqcup \pi_2$ nécessite également quelques précautions : l'approche naïve $f(\pi_1 \sqcup \pi_2) = \pi_1 \cup \pi_2 \cup (\pi_1 \sqcap \pi_2)$ conduirait à une croissance exponentielle. Notre traduction réutilise la technique de copie introduite ci-dessus.

Nous étendons la copie des variables aux ensembles de variables et aux programmes. Nous associons à chaque ensemble de variables $P \subseteq \mathbb{P}$ les copies $(P)^1 = \{(p)^1 : p \in P\}$ et $(P)^2 = \{(p)^2 : p \in P\}$; et nous associons les programmes π à $(\pi)^1$ en remplaçant toutes les variables p de π par $(p)^1$; de même, nous associons π à $(\pi)^2$. Par exemple, $(p{\leftarrow}q \wedge r)^2 = (p)^2{\leftarrow}(q)^2 \wedge (r)^2$.

Proposition 6. *Soit $V, U, W \subseteq \mathbb{P}_\pi$ et soit k égal à 1 ou 2. Alors :*

- *$(V, U, W) \in \|\pi\|$ ssi $((V)^k, (U)^k, (W)^k) \in \|(\pi)^k\|$;*
- *$V \in \|\varphi\|$ ssi $(V)^k \in \|(\varphi)^k\|$.*

La preuve se fait par induction sur la structure de π et φ.

3.1 La traduction

Nous allons montrer que $\sqcap$ et $\sqcup$ peut être polynomialement éliminé des formules DL-PPA. La formule résultante est dans le langage de DL-PA, ce qui nous permettra de transférer les résultats de complexité. La définition récursive de la traduction des formules et des programmes est donnée dans la Table 2. La traduction est homomorphe pour tous les opérateurs de programme et opérateurs logiques de PDL, nous ne commentons donc que les cas non triviaux : affectations, composition parallèle et composition non déterministe inclusive.

- Chaque affectation $p{\leftarrow}\varphi$ est traduite en stockant en plus que p a été affecté (via une nouvelle variable δ_p).
- La traduction de la composition parallèle $\pi_1 \sqcap \pi_2$ repose sur l'introduction de deux nouvelles copies $(p)^1$ et $(p)^2$ de chaque variable propositionnelle p qui intervient dans $\pi_1 \sqcap \pi_2$. Puis pour chaque programme π, le programme $(\pi)^1$ est obtenu en substituant chaque p dans π par sa copie $(p)^1$; de même pour $(\pi)^2$. La traduction de $\pi_1 \sqcap \pi_2$ commence en faisant des copies locales pour chaque sous-programme, exécute ensuite π_1 et π_2 sur ces copies, et vérifie ensuite si les deux résultats peuvent être fusionnés : est-ce que les variables qui ont été assignées par les deux sous-programmes modifient leurs copies de la même manière, c'est-à-dire que $(\delta_{(p)^1} \wedge \delta_{(p)^2}) \rightarrow ((p)^1 \leftrightarrow (p)^2)$ pour chaque $p \in \mathbb{P}_{\pi_1} \cap \mathbb{P}_{\pi_2}$? Si c'est le cas, les valeurs des copies pertinentes sont recopiées dans chaque p (nous ne faisons rien lorsqu'aucune copie n'a été affectée). Enfin, les variables auxiliaires $(p)^1$ et $(p)^2$ sont définies à la valeur faux.
- Dans la traduction d'une composition non déterministe inclusive $\pi_1 \sqcup \pi_2$, le contrôle de compatibilité suivi d'une recopie est remplacé par une composition non déterministe exhaustive de : soit une vérification de la compatibilité et une recopie des valeurs calculées par $\pi_1 \sqcap \pi_2$, soit une recopie des valeurs calculées par π_1, soit une recopie les valeurs calculées par π_2.

Par exemple, $f(p{\leftarrow}\top \sqcap p{\leftarrow}\bot)$ est

$$(p)^1{\leftarrow}p; (p)^2{\leftarrow}p; (p)^1{\leftarrow}\top; \delta_{(p)^1}{\leftarrow}\top; (p)^2{\leftarrow}\bot; \delta_{(p)^2}{\leftarrow}\top;$$
$$(\delta_{(p)^1} \wedge \delta_{(p)^2}) \rightarrow ((p)^1 \leftrightarrow (p)^2)?;$$
$$((\delta_{(p)^1}?; p{\leftarrow}(p)^1) \cup (\delta_{(p)^2}?; p{\leftarrow}(p)^2) \cup \neg\delta_{(p)^1} \wedge \neg\delta_{(p)^2}?);$$
$$\delta_{(p)^1}{\leftarrow}\bot; \delta_{(p)^2}{\leftarrow}\bot; (p)^1{\leftarrow}\bot; (p)^2{\leftarrow}\bot$$

Comme le programme d'origine, le programme traduit a une interprétation vide car après après avoir défini $\delta_{(p)^2}$ et $\delta_{(p)^1}$ à la valeur vrai et $(p)^1$ à la valeur vrai et $(p)^2$ à la valeur faux, le test $(\delta_{(p)^1} \wedge \delta_{(p)^2}) \rightarrow ((p)^1 \leftrightarrow (p)^2)?$ va échouer.

3.2 Correction de la traduction

Lemme 1. *Soient $V, U, W \subseteq \mathbb{P}_{\pi_1 \sqcap \pi_2}$ des valuations. Soit $\delta_W = \{\delta_p : p \in W\}$. Alors $(V, U \cup \delta_W) \in \|f(\pi_1 \sqcap \pi_2)\|_{\text{DL-PA}}$ si et seulement si il existe U_1, W_1, U_2, W_2 tels que :*

1. *$(V, U_1 \cup \delta_{W_1}) \in \|f(\pi_1)\|_{\text{DL-PA}}$;*
2. *$(V, U_2 \cup \delta_{W_2}) \in \|f(\pi_2)\|_{\text{DL-PA}}$;*
3. *$U_1 \cap W_1 \cap W_2 = U_2 \cap W_1 \cap W_2$;*
4. *$U = (V \setminus W) \cup (U_1 \cap W_1) \cup (U_2 \cap W_2)$;*
5. *$W = W_1 \cup W_2$.*

Lemme 2. *Soit $V, U, W \subseteq \mathbb{P}_{\pi_1 \sqcup \pi_2}$. Soit $\delta_W = \{\delta_p : p \in W\}$. Alors $(V, U \cup \delta_W) \in \|f(\pi_1 \sqcup \pi_2)\|_{\text{DL-PA}}$ si et seulement si $(V, U \cup \delta_W) \in \|f(\pi_1)\|_{\text{DL-PA}}$ ou $(V, U \cup \delta_W) \in \|f(\pi_2)\|_{\text{DL-PA}}$ ou $(V, U \cup \delta_W) \in \|f(\pi_1 \sqcap \pi_2)\|_{\text{DL-PA}}$.*

Proposition 7. *Soit $V, U, W \subseteq \mathbb{P}_{\pi_1 \sqcup \pi_2}$. Soit $\delta_W = \{\delta_p : p \in W\}$. Alors :*

$$
\begin{array}{ll}
f(p) = p & f(p{\leftarrow}\varphi) = p{\leftarrow}f(\varphi); \delta_p{\leftarrow}\top \\
f(\bot) = \bot & f(\varphi?) = f(\varphi)? \\
f(\neg\varphi) = \neg f(\varphi) & f(\pi_1;\pi_2) = f(\pi_1); f(\pi_2) \\
f(\varphi_1 \vee \varphi_2) = f(\varphi_1) \vee f(\varphi_2) & f(\pi_1 \cup \pi_2) = f(\pi_1) \cup f(\pi_2) \\
f(\langle\pi\rangle\varphi) = \langle f(\pi)\rangle f(\varphi) & f(\pi^*) = (f(\pi))^*
\end{array}
$$

$$
\begin{array}{l}
f(\pi_1 \sqcap \pi_2) = \Big(;_{p\in\mathbb{P}_{\pi_1}} (p)^1{\leftarrow}p\Big); \Big(;_{p\in\mathbb{P}_{\pi_2}} (p)^2{\leftarrow}p\Big); f\Big((\pi_1)^1\Big); f\Big((\pi_2)^2\Big); \Big(\bigwedge_{p\in\mathbb{P}_{\pi_1}\cap\mathbb{P}_{\pi_2}} ((\delta_{(p)^1} \wedge \delta_{(p)^2}) \rightarrow ((p)^1 \leftrightarrow (p)^2))\Big)?; \\
\qquad \Big(;_{p\in\mathbb{P}_{\pi_1\sqcap\pi_2}} (\delta_{(p)^1}?; p{\leftarrow}(p)^1) \cup (\delta_{(p)^2}?; p{\leftarrow}(p)^2) \cup \neg\delta_{(p)^1} \wedge \neg\delta_{(p)^2}?\Big); \Big(;_{p\in\mathbb{P}_{\pi_1\sqcap\pi_2}} (\delta_{(p)^1}{\leftarrow}\bot; \delta_{(p)^2}{\leftarrow}\bot; (p)^1{\leftarrow}\bot; (p)^2{\leftarrow}\bot)\Big) \\
f(\pi_1 \sqcup \pi_2) = \Big(;_{p\in\mathbb{P}_{\pi_1}} (p)^1{\leftarrow}p\Big); \Big(;_{p\in\mathbb{P}_{\pi_2}} (p)^2{\leftarrow}p\Big); f\Big((\pi_1)^1\Big); f\Big((\pi_2)^2\Big); \\
\qquad \Big(\Big(\bigwedge_{p\in\mathbb{P}_{\pi_1}\cap\mathbb{P}_{\pi_2}} ((\delta_{(p)^1} \wedge \delta_{(p)^2}) \rightarrow ((p)^1 \leftrightarrow (p)^2))\Big)?; \Big(;_{p\in\mathbb{P}_{\pi_1\sqcap\pi_2}} (\delta_{(p)^1}?; p{\leftarrow}(p)^1) \cup (\delta_{(p)^2}?; p{\leftarrow}(p)^2) \cup \neg\delta_{(p)^1} \wedge \neg\delta_{(p)^2}?\Big)\Big) \\
\qquad \cup \Big(;_{p\in\mathbb{P}_{\pi_1}} p{\leftarrow}(p)^1\Big) \cup \Big(;_{p\in\mathbb{P}_{\pi_2}} p{\leftarrow}(p)^2\Big)\Big); \Big(;_{p\in\mathbb{P}_{\pi_1\sqcap\pi_2}} (\delta_{(p)^1}{\leftarrow}\bot; \delta_{(p)^2}{\leftarrow}\bot; (p)^1{\leftarrow}\bot; (p)^2{\leftarrow}\bot)\Big)
\end{array}
$$

TABLE 2 – Traduction de DL-PPA à DL-PA

1. $\|\varphi\| = \|f(\varphi)\|_{\text{DL-PA}}$;
2. $(V, U, W) \in \|\pi\|$ *ssi* $(V, U \cup \delta_W) \in \|f(\pi)\|_{\text{DL-PA}}$.

Démonstration. La preuve se fait par induction sur la forme des formules et des programmes.
Pour le cas de la composition parallèle, nous utilisons le lemme 1 et pour la composition non déterministe inclusive, nous utilisons le lemme 2.
Dans le cas d'une formule de la forme $\langle\pi\rangle\varphi$ nous avons : $V \in \|\langle\pi\rangle\varphi\|$ ssi il existe U et W tels que $(V, U, W) \in \|\pi\|$ et $U \in \|\varphi\|$. Cette dernière est par hypothèse d'induction équivalente à : il existe U et W tels que $(V, U\cup\delta_W) \in \|f(\pi)\|_{\text{DL-PA}}$ et $U \in \|f(\varphi)\|_{\text{DL-PA}}$. Ce qui par la Proposition 2 est équivalent à : $(V, U\cup\delta_W) \in \|f(\pi)\|_{\text{DL-PA}}$ et $U\cup\delta_W \in \|f(\varphi)\|_{\text{DL-PA}}$, c'est-à-dire, à $V \in \|\langle f(\pi)\rangle f(\varphi)\|_{\text{DL-PA}} = \|f(\langle\pi\rangle\varphi)\|_{\text{DL-PA}}$. □

Notons que le deuxième élément concerne uniquement les évaluations composées de variables qui interviennent dans π. Cela évite de traiter les valeurs initiales (arbitraires) des variables auxiliaires $(p)^1$ et $(p)^2$. Notre restriction est justifiée par la Proposition 2.
La traduction de DL-PPA vers DL-PA est une transformation polynomiale :

Proposition 8. *La longueur $\ell(f(\varphi))$ de la traduction d'une formule φ de* DL-PPA *est polynôme de la longueur $\ell(\varphi)$ de φ.*

3.3 Résultats de complexité

Nous utilisons la traduction afin d'établir des limites supérieures de complexité. Les limites inférieures sont importées du fragment DL-PA.

Théorème 1. *Le problème de vérification de modèle* DL-PPA *de décider, étant donné V et φ, si $V \in \|\varphi\|$ est PSPACE-complet.*

Démonstration. La borne inférieure est due au fait que le problème de vérification de modèle DL-PA est PSPACE-dur [Herzig et al., 2011]. La borne supérieure est obtenue par la Proposition 7 et la proposition 8 grâce à l'appartenance de DL-PA à PSPACE [Balbiani et al., 2014]. □

Pour établir la complexité de la vérification de satisfiabilité nous la réduisons polynomialement à la vérification de modèle.

Proposition 9. *Une formule φ de* DL-PPA *est satisfiable ssi $\emptyset \in \|\langle;_{p\in\mathbb{P}_\varphi} (p{\leftarrow}\top \cup p{\leftarrow}\bot)\rangle\varphi\|$.*

Théorème 2. *Le problème de vérification de la satisfiabilité* DL-PPA *est PSPACE-complet.*

Démonstration. La borne inférieure est due au fait que le problème de vérification de satisfiabilité de DL-PA est PSPACE-dur [Herzig et al., 2011]. La borne supérieure découle de l'appartenance de la vérification de modèle DL-PPA à PSPACE via la Proposition 9. □

4 Application de DL-PPA à la planification

Dans cette section, nous définissons formellement les actions et les tâches de planification séquentielle et parallèle dans notre cadre DL-PPA. Nous montrons ensuite qu'une vérification de modèle DL-PPA peut être utilisée pour tester la solvabilité de telles tâches.

4.1 Planification séquentielle avec effets conditionnels

Une tâche de planification est un triplet $(Act, V_0, Goal)$ où :
- *Act* est un ensemble d'*actions conditionnelles* ;
- $V_0 \subseteq \mathbb{P}$ est une valuation (l'*état initial*) ;
- $Goal \in Fml_{\text{DL-PPA}}$ est le *but*.

Une action conditionnelle est une paire $\mathsf{a} = (pre(\mathsf{a}), eff(\mathsf{a}))$ où :
- $pre(\mathsf{a}) \in Fml_{\text{DL-PPA}}$ est la *précondition* de a ;
- $eff(\mathsf{a}) \in Fml_{\text{DL-PPA}} \times 2^{\mathbb{P}} \times 2^{\mathbb{P}}$ est un ensemble de triplets ce de la forme $(cnd(ce), ceff^+(ce), ceff^-(ce))$, les *effets conditionnels* de a, où $cnd(ce)$ est une

formule de DL-PPA (la condition) et $\mathit{ceff}^{+}(ce)$ et $\mathit{ceff}^{-}(ce)$ sont des ensembles de variables qui sont respectivement ajoutées ou supprimées par a si la condition ce est vraie.

Nous disons qu'un état V est *accessible par un plan séquentiel* à partir d'un état V_0 via un ensemble d'actions conditionnelles Act s'il y a un *plan séquentiel*, c'est-à-dire, une séquence d'actions $\langle \mathsf{a}_1, \ldots, \mathsf{a}_m \rangle$ de Act, et une séquence d'états $\langle V_0, \ldots, V_m \rangle$ avec $m \geq 0$ tel que $V = V_m$ et $\tau_{\mathsf{a}_k}(V_{k-1}) = V_k$ pour chaque k tel que $1 \leq k \leq m$. Une tâche de planification est *résoluble par un plan séquentiel* s'il y a au moins un état $V \in \|\mathit{Goal}\|$ tel que V est accessible par un plan séquentiel à partir de V_0 via Act ; sinon, il est insoluble par un plan séquentiel.

Une action conditionnelle a détermine une fonction partielle τ_{a} de $2^{\mathbb{P}}$ dans $2^{\mathbb{P}}$: $\tau_{\mathsf{a}}(V)$ est définie (a est exécutable dans l'état V) si et seulement si $V \in \|\mathit{pre}(\mathsf{a})\|$ et pour tout $ce_1, ce_2 \in \mathit{eff}(\mathsf{a})$ tel que $V \in \|\mathit{cnd}(ce_1) \wedge \mathit{cnd}(ce_2)\|$: $(\mathit{ceff}^{+}(ce_1) \cap \mathit{ceff}^{-}(ce_2)) \cup (\mathit{ceff}^{-}(ce_1) \cap \mathit{ceff}^{+}(ce_2)) = \emptyset$. Quand τ_{a} est définie dans V alors :

$$\tau_{\mathsf{a}}(V) = \left(V \setminus \bigcup_{\substack{ce \in \mathit{eff}(\mathsf{a}) \\ V \in \|\mathit{cnd}(ce)\|}} \mathit{ceff}^{-}(ce) \right) \cup \bigcup_{\substack{ce \in \mathit{eff}(\mathsf{a}) \\ V \in \|\mathit{cnd}(ce)\|}} \mathit{ceff}^{+}(ce)$$

Nous associons à l'action a le programme DL-PPA $\mathsf{exeAct}(\mathsf{a})$:

$$\mathsf{exeAct}(\mathsf{a}) = \mathit{pre}(\mathsf{a})? \sqcap \bigsqcap_{ce \in \mathit{eff}(\mathsf{a})} \left(\begin{array}{l} \neg \mathit{cnd}(ce)? \\ \cup \begin{pmatrix} \mathit{cnd}(ce)? \\ \sqcap \bigsqcap_{p \in \mathit{ceff}^{+}(ce)} p \leftarrow \top \\ \sqcap \bigsqcap_{q \in \mathit{ceff}^{-}(ce)} q \leftarrow \bot \end{pmatrix} \end{array} \right)$$

Le programme $\mathsf{exeAct}(\mathsf{a})$ se comporte comme a :

Lemme 3. *Pour chaque action* $\mathsf{a} \in Act$ *:*

1. τ_{a} *est définie dans* V *ssi il existe* U, W *tels que* $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{a})\|$ *;*
2. *Si* τ_{a} *est définie dans* V *alors* $\tau_{\mathsf{a}}(V) = U$ *ssi* $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{a})\|$ *pour un certain* W*.*

Démonstration. Prenons un état arbitraire V. $\tau_{\mathsf{a}}(V)$ n'est pas définie ssi : (1) $V \notin \|\mathit{pre}(\mathsf{a})\|$, alors le programme échoue parce que $\|\mathit{pre}(\mathsf{a})?\| = \emptyset$, ou (2) il existe $ce_1, ce_2 \in \mathit{eff}(\mathsf{a})$ et $p \in \mathbb{P}$ tels que $V \in \|\mathit{cnd}(ce_1) \wedge \mathit{cnd}(ce_2)\|$ et $(\mathit{ceff}^{+}(ce_1) \cap \mathit{ceff}^{-}(ce_2)) \cup (\mathit{ceff}^{-}(ce_1) \cap \mathit{ceff}^{+}(ce_2))$ contient un p, alors le programme échoue parce que $\|p \leftarrow \bot \sqcap p \leftarrow \top\| = \emptyset$. Quand $\tau_{\mathsf{a}}(V)$ est définie alors $V \in \|\mathit{pre}(\mathsf{a})\|$, de sorte que $(V, V, \emptyset) \in \|\mathit{pre}(\mathsf{a})?\|$. De plus, pour chaque $ce \in \mathit{eff}(\mathsf{a})$ tel que $V \in \|\mathit{cnd}(ce)\|$ les programmes $(\bigsqcap_{p \in \mathit{ceff}^{+}(ce)} p \leftarrow \top)$ et $(\bigsqcap_{q \in \mathit{ceff}^{-}(ce)} q \leftarrow \bot)$ sont exécutés en parallèle et toutes les affectations sont consistantes (aucun $p \leftarrow \bot$ et $p \leftarrow \top$ n'est exécuté en parallèle). Alors, la composition parallèle de tous ces programmes conduit, par définition, à l'état $\tau_{\mathsf{a}}(V) = U$, avec $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{a})\|$, où U est l'ensemble des variables affectées à $\top$ et W l'ensemble de toutes les variables affectées dans le programme $\mathsf{exeAct}(\mathsf{a})$. □

4.2 Planification parallèle avec effets conditionnels

Afin de définir la solvabilité d'une tâche de planification par un plan parallèle, nous devons déterminer les conditions d'exécutabilité parallèle d'un ensemble d'actions conditionnelles $A = \{\mathsf{a}_1, \ldots, \mathsf{a}_m\}$ dans un état V. Nous allons suivre la sémantique $\forall$-step et la notion d'interférence dans un état de [Rintanen et al., 2006]. En particulier, pour que la vérification de plan reste polynomiale, nous considérons que deux actions sont indépendantes et peuvent être exécutées en parallèle si ces actions n'ont pas d'effets contradictoires ni d'interactions croisées [Rintanen et al., 2006, Theorem 2.12]. D'une part, nous disons que deux actions a et a' qui sont exécutables dans V ont *des effets contradictoires dans l'état* V ssi il y a des effets conditionnels $ce \in \mathit{eff}(\mathsf{a})$ et $ce' \in \mathit{eff}(\mathsf{a}')$ tels que :

- $V \in \|\mathit{cnd}(ce)\|$ et $V \in \|\mathit{cnd}(ce')\|$, et
- $\mathit{ceff}^{+}(ce) \cap \mathit{ceff}^{-}(ce') \neq \emptyset$ ou $\mathit{ceff}^{-}(ce) \cap \mathit{ceff}^{+}(ce') \neq \emptyset$.

D'autre part, nous disons que l'action a a une interaction croisée avec $\mathsf{a}' \neq \mathsf{a}$ dans l'état V ssi s'il existe un effet conditionnel $ce' \in \mathit{eff}(\mathsf{a}')$ tel que :

- $V \in \|\mathit{cnd}(ce')\|$, et
- $V \notin \|\langle (\bigsqcap_{p \in \mathit{ceff}^{+}(ce')} p \leftarrow \top) \sqcap (\bigsqcap_{p \in \mathit{ceff}^{-}(ce')} p \leftarrow \bot) \rangle \varphi\|$ où $\varphi = \mathit{pre}(\mathsf{a}) \wedge \left(\bigwedge_{\substack{ce \in \mathit{eff}(\mathsf{a}) \\ V \in \|\mathit{cnd}(ce)\|}} \mathit{cnd}(ce) \right)$.

Nous disons que a et a' ont des *interactions croisées dans l'état* V si a a une interaction croisée avec a' dans l'état V ou a' a une interaction croisée avec a dans l'état V.

Autrement dit, dans tout plan parallèle, aucun effet d'une action ne peut être détruit par un effet d'une autre action exécutée en parallèle, et aucune précondition ou condition d'un effet conditionnel d'une action ne peut être détruite par l'effet d'une autre action exécutée en parallèle. Par exemple, en planification classique parallèle, il n'est pas possible de saisir et déposer un même objet en parallèle (effets contradictoires) et il n'est pas possible pour deux agents de ramasser le même objet en parallèle (interactions croisées).

Un ensemble d'actions conditionnelles $\mathsf{A} = \{\mathsf{a}_1, \ldots, \mathsf{a}_m\}$ détermine une fonction partielle τ_{A} de $2^{\mathbb{P}}$ dans $2^{\mathbb{P}}$: $\tau_{\mathsf{A}}(V)$ est définie ($\mathsf{a}_1, \ldots, \mathsf{a}_m$ sont exécutables en parallèle dans l'état V) si et seulement si :

1. pour toute $\mathsf{a}_i \in \mathsf{A}$, τ_{a_i} est définie dans V, et
2. pour toute $\mathsf{a}_i, \mathsf{a}_j \in \mathsf{A}$ telle que $i \neq j$:
   - a_i et a_j n'ont pas d'effets contradictoires, et
   - a_i et a_j n'ont pas d'interactions croisées.

Quand τ_{A} est définie dans V alors :

$$\tau_{\mathsf{A}}(V) = \left(V \setminus \bigcup_{\substack{\mathsf{a} \in \mathsf{A}, ce \in \mathit{eff}(\mathsf{a}), \\ V \in \|\mathit{cnd}(ce)\|}} \mathit{ceff}^{-}(ce) \right) \cup \bigcup_{\substack{\mathsf{a} \in \mathsf{A}, ce \in \mathit{eff}(\mathsf{a}), \\ V \in \|\mathit{cnd}(ce)\|}} \mathit{ceff}^{+}(ce)$$

À chaque ensemble d'actions conditionnelles, nous pouvons associer un programme DL-PPA qui se comporte exactement comme l'exécution parallèle de ses éléments.

Étant donné $\mathsf{A} = \{\mathsf{a}_1, \ldots, \mathsf{a}_m\}$, soit $\mathsf{exeAct}(\mathsf{A})$ le programme DL-PPA :

$$\bigsqcap_{\mathsf{a}\in\mathsf{A}} \left(\begin{array}{l} \mathsf{exeAct}(\mathsf{a}) \sqcap \bigsqcap_{\substack{\mathsf{a}'\in\mathsf{A}\\ \mathsf{a}\neq\mathsf{a}'}} \langle \mathsf{exeAct}(\mathsf{a}')\rangle pre(\mathsf{a})? \\ \sqcap \bigsqcap_{\substack{\mathsf{a}'\in\mathsf{A}\\ \mathsf{a}\neq\mathsf{a}'}} \bigsqcap_{ce\in eff(\mathsf{a})} (cnd(ce) \rightarrow \langle \mathsf{exeAct}(\mathsf{a}')\rangle cnd(ce))? \end{array} \right)$$

Lemme 4. *Pour tout ensemble fini d'actions* $\mathsf{A} = \{\mathsf{a}_1, \ldots, \mathsf{a}_m\}$,

1. τ_{A} *est définie dans* V *ssi il existe* U, W *tels que* $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{A})\|$ *;*
2. *Si* τ_{A} *est définie dans* V *alors* $\tau_{\mathsf{A}}(V) = U$ *ssi* $(V, U, W) \in \|\mathsf{exeAct}(\mathsf{A})\|$ *pour un certain* W.

Démonstration. Prenons un état arbitraire V. Par le Lemme 3, pour tout $i \in \{1, \ldots, m\}$ nous savons que $\mathsf{exeAct}(\mathsf{a}_i)$ se comporte comme a_i, et donc est exécutable si $\tau_{\mathsf{a}_i}(V)$ est définie. Si deux actions $\mathsf{a}, \mathsf{a}' \in \mathsf{A}$ ont des effets contradictoires dans l'état V, il existe $ce \in eff(\mathsf{a})$ et $ce' \in eff(\mathsf{a}')$ et $p \in \mathbb{P}$ tels que $V \in \|cnd(ce) \wedge cnd(ce')\|$ et $(ceff^{+}(ce) \cap ceff^{-}(ce')) \cup (ceff^{-}(ce) \cap ceff^{+}(ce'))$ contient p, alors le programme échoue à cause de l'exécution de la composition parallèle de $p{\leftarrow}\bot$ et $p{\leftarrow}\top$ dans le programme $\mathsf{exeAct}(\mathsf{a}) \sqcap \mathsf{exeAct}(\mathsf{a}')$. Si l'action a a une interaction croisée avec l'action a' dans l'état V, il existe un effet confitionnel $ce' \in eff(\mathsf{a}')$ tel que $V \in \|cnd(ce')\|$ et, soit $V \notin \|\langle(\bigsqcap_{p\in ceff^{+}(ce')} p{\leftarrow}\top) \sqcap (\bigsqcap_{p\in ceff^{-}(ce')} p{\leftarrow}\bot)\rangle pre(\mathsf{a})\|$, et donc l'exécution de $\langle \mathsf{exeAct}(\mathsf{a}')\rangle pre(\mathsf{a})?$ échoue, soit il existe un effet conditionnel $ce \in eff(\mathsf{a})$ tel que $V \in \|cnd(ce)\|$ et $V \notin \|\langle(\bigsqcap_{p\in ceff^{+}(ce')} p{\leftarrow}\top) \sqcap (\bigsqcap_{p\in ceff^{-}(ce')} p{\leftarrow}\bot)\rangle cnd(ce)\|$, et donc l'exécution de $(cnd(ce) \rightarrow \langle \mathsf{exeAct}(\mathsf{a}')\rangle cnd(ce))?$ échoue. Enfin, la composition parallèle de tous ces programmes vérifie les effets contradictoires et les interactions croisées et conduit à l'état $\tau_{\mathsf{A}}(V)$ par définition en raison de l'exécution parallèle de tous les $\mathsf{exeAct}(\mathsf{a}_i)$. □

Nous disons qu'un état V est *accessible par un plan parallèle* à partir d'un état V_0 via un ensemble d'actions conditionnelles Act s'il y a un *plan parallèle*, c'est-à-dire, une séquence $\langle \mathsf{A}_1, \ldots, \mathsf{A}_m\rangle$ d'ensembles d'actions $\mathsf{A}_i \subseteq Act$ et une séquence d'états $\langle V_0, \ldots, V_m\rangle$ avec $m \geq 0$ tels que $V = V_m$ et $\tau_{\mathsf{A}_k}(V_{k-1}) = V_k$ pour chaque k tel que $1 \leq k \leq m$. Une tâche de planification $(Act, V_0, Goal)$ est *résoluble par un plan parallèle* s'il y a au moins un état $V \in \|Goal\|$ tel que V est accessible par un plan parallèle à partir de V_0 via Act ; sinon, il est insoluble par un plan parallèle.

4.3 Solvabilité des tâches de planification avec horizon borné

Maintenant que nous avons défini un codage parallèle des actions et la solvabilité d'une tâche de planification, nous pouvons capturer la solvabilité d'une tâche de planification dans DL-PPA avec un horizon borné k. Ce problème est connu pour être PSPACE-complet [Bylander, 1994].

Théorème 3. *Une tâche de planification* $(Act, V_0, Goal)$ *est résoluble par un plan séquentiel avec pas plus de k actions si et seulement si :*

$$V_0 \in \left\| \left\langle \left(\bigcup_{\mathsf{a}\in Act} \mathsf{exeAct}(\mathsf{a}) \right)^{\leq k} \right\rangle Goal \right\|$$

Démonstration. Notre formule se lit "il existe une exécution de $(\bigcup_{\mathsf{a}\in Act} \mathsf{exeAct}(\mathsf{a}))^{\leq k}$ après laquelle *Goal* est vrai." Nous savons parle Lemme 3 que $\mathsf{exeAct}(\mathsf{a})$ se comporte correctement et produit les mêmes effets que l'action a. Le programme $(\bigcup_{\mathsf{a}\in Act} \mathsf{exeAct}(\mathsf{a}))^{\leq k}$ choisit de manière non déterministe une action a de Act et exécute le programme correspondant $\mathsf{exeAct}(\mathsf{a})$, et répète ensuite ceci un nombre de fois inférieur ou égal à k. Cela produit une séquence d'au plus k actions, c'est-à-dire un plan séquentiel borné par k. Par conséquent, la formule est satisfaite dans l'état initial si et seulement s'il existe un plan séquentiel de longueur bornée par k après lequel l'objectif est satisfait, c'est-à-dire, si et seulement si la tâche de planification est résoluble avec une séquence d'au plus k actions. □

Théorème 4. *Une tâche de planification* $(Act, V_0, Goal)$ *est résoluble par un plan parallèle avec pas plus de k étapes si et seulement si :*

$$V_0 \in \left\| \left\langle \left(\bigsqcap_{\mathsf{a}\in Act} exe_{\mathsf{a}}{\leftarrow}\bot ; \bigsqcup_{\mathsf{a}\in Act} exe_{\mathsf{a}}{\leftarrow}\top ; \pi_{exe} \right)^{\leq k} \right\rangle Goal \right\|$$

où $exe_{\mathsf{a}} \notin \mathbb{P}_{\mathsf{exeAct}(\mathsf{a})}$ *pour toute* $\mathsf{a} \in Act$, *et*

$$\pi_{exe} = \bigsqcap_{\mathsf{a}\in Act} \left(\begin{array}{l} \neg exe_{\mathsf{a}}? \cup \\ \left(\begin{array}{l} exe_{\mathsf{a}}? \sqcap \mathsf{exeAct}(\mathsf{a}) \\ \sqcap \bigsqcap_{\substack{\mathsf{a}'\in Act\\ \mathsf{a}\neq\mathsf{a}'}} \left(\begin{array}{l} \neg exe_{\mathsf{a}'}? \\ \cup \left(\begin{array}{l} exe_{\mathsf{a}'}? \sqcap \langle \mathsf{exeAct}(\mathsf{a}')\rangle pre(\mathsf{a})? \\ \sqcap \bigsqcap_{ce\in eff(\mathsf{a})} (cnd(ce) \rightarrow \langle \mathsf{exeAct}(\mathsf{a}')\rangle cnd(ce))? \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right)$$

Démonstration. Le programme $\bigsqcap_{\mathsf{a}\in Act} exe_{\mathsf{a}}{\leftarrow}\bot$ initialise une nouvelle variable fraîche spéciale $exe_{\mathsf{a}} \notin \mathbb{P}_{\mathsf{exeAct}(\mathsf{a})}$ à la valeur $\bot$, pour chaque action $\mathsf{a} \in Act$. Ensuite, la composition non déterministe inclusive $\bigsqcup_{\mathsf{a}\in Act} exe_{\mathsf{a}}{\leftarrow}\top$ choisit un sous-ensemble d'actions non vide $A \subseteq Act$ et exécute le programme $\bigsqcap_{\mathsf{a}\in A} exe_{\mathsf{a}}{\leftarrow}\top$. À ce point, $exe_{\mathsf{a}} = \top$ ssi $\mathsf{a} \in \mathsf{A}$, et le programme π_{exe} est exécuté. Nous pouvons voir facilement que, pour un ensemble d'actions choisi A, π_{exe} se comporte comme le programme $\mathsf{exeAct}(\mathsf{A})$. Nous savons par le Lemme 4 que ce dernier programme se comporte correctement et produit le même effet que l'exécution parallèle de toutes les actions dans A. La séquence $\bigsqcap_{\mathsf{a}\in Act} exe_{\mathsf{a}}{\leftarrow}\bot ; \bigsqcup_{\mathsf{a}\in Act} exe_{\mathsf{a}}{\leftarrow}\top ; \pi_{exe}$ est ensuite répétée un nombre de fois inférieur ou égal à k. Cela produit une séquence d'au plus k exécutions parallèles d'ensembles d'actions, c'est-à-dire un plan parallèle borné par k. Par conséquent, la formule est satisfaite dans l'état initial si et seulement s'il existe un plan parallèle de longueur bornée par k après lequel le but est satisfait, c'est-à-dire, si et seulement si la tâche de planification est résoluble avec une séquence d'au plus k étapes parallèles. □

Dans le théorème 3 et dans le théorème 4, la taille des problèmes de vérification de modèle est polynomiale dans la taille de la tâche de planification plus log k. Cela repose sur notre représentation compacte des programmes bornés $\pi^{\leq k}$, cf. Proposition 5.

5 Conclusion

Nous avons montré comment capturer la planification classique parallèle dans une extension de DL-PA par une composition non déterministe parallèle et inclusive dont les problèmes de vérification de modèle et de vérification de satisfiabilité sont toujours dans PSPACE. Cela permet notamment de décider, dans ces limites de complexité, de l'existence d'un plan à horizon fini.
Une poursuite directe de notre travail sera la planification épistémique parallèle. Nous pouvons prendre en charge l'extension épistémique de DL-PA en termes de variables d'observation qui a déjà été appliquée à la planification épistémique séquentielle dans [Cooper et al., 2016].

Références

[Andersen et al., 2012] Andersen, M. B., Bolander, T., and Jensen, M. H. (2012). Conditional epistemic planning. In *Logics in Artificial Intelligence - 13th European Conference, JELIA*, pages 94–106.

[Balbiani and Boudou, 2018] Balbiani, P. and Boudou, J. (2018). Iteration-free PDL with storing, recovering and parallel composition : a complete axiomatization. *J. Log. Comput.*, 28(4) :705–731.

[Balbiani et al., 2014] Balbiani, P., Herzig, A., Schwarzentruber, F., and Troquard, N. (2014). DL-PA and DCL-PC : model checking and satisfiability problem are indeed in PSPACE. *CoRR*, abs/1411.7825.

[Balbiani et al., 2013] Balbiani, P., Herzig, A., and Troquard, N. (2013). Dynamic logic of propositional assignments : A well-behaved variant of PDL. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 143–152.

[Balbiani and Vakarelov, 2003] Balbiani, P. and Vakarelov, D. (2003). PDL with intersection of programs : A complete axiomatization. *Journal of Applied Non-Classical Logics*, 13(3-4) :231–276.

[Benevides et al., 2011] Benevides, M. R. F., de Freitas, R. P., and Viana, J. P. (2011). Propositional dynamic logic with storing, recovering and parallel composition. *Electr. Notes Theor. Comput. Sci.*, 269 :95–107.

[Benevides and Schechter, 2014] Benevides, M. R. F. and Schechter, L. M. (2014). Propositional dynamic logics for communicating concurrent programs with ccs's parallel operator. *J. Log. Comput.*, 24(4) :919–951.

[Blum and Furst, 1997] Blum, A. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2) :281–300.

[Bolander et al., 2018] Bolander, T., Engesser, T., Mattmüller, R., and Nebel, B. (2018). Better eager than lazy ? how agent types impact the successfulness of implicit coordination. In *Principles of Knowledge Representation and Reasoning : Proceedings of the Sixteenth International Conference, KR 2018*, pages 445–453.

[Bylander, 1994] Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2) :165–204.

[Cong et al., 2018] Cong, S. L., Pinchinat, S., and Schwarzentruber, F. (2018). Small undecidable problems in epistemic planning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 4780–4786.

[Cooper et al., 2016] Cooper, M. C., Herzig, A., Maffre, F., Maris, F., and Régnier, P. (2016). A simple account of multi-agent epistemic planning. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, pages 193–201.

[Dimopoulos et al., 1997] Dimopoulos, Y., Nebel, B., and Koehler, J. (1997). Encoding planning problems in nonmonotonic logic programs. In *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997, Proceedings*, pages 169–181.

[Goldblatt, 1992] Goldblatt, R. (1992). Parallel action : Concurrent dynamic logic with independent modalities. *Studia Logica*, 51(3/4) :551–578.

[Herzig, 2014] Herzig, A. (2014). Belief change operations : A short history of nearly everything, told in dynamic logic of propositional assignments. In *Principles of Knowledge Representation and Reasoning : Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*.

[Herzig et al., 2014] Herzig, A., de Menezes, M. V., de Barros, L. N., and Wassermann, R. (2014). On the revision of planning tasks. In *ECAI 2014 - 21st European Conference on Artificial Intelligence*, pages 435–440.

[Herzig et al., 2011] Herzig, A., Lorini, E., Moisan, F., and Troquard, N. (2011). A dynamic logic of normative systems. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 228–233.

[Knoblock, 1994] Knoblock, C. A. (1994). Generating parallel execution plans with a partial-order planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 98–103.

[Li et al., 2017] Li, Y., Yu, Q., and Wang, Y. (2017). More for free : a dynamic epistemic framework for conformant planning over transition systems. *J. Log. Comput.*, 27(8) :2383–2410.

[Mayer and Stockmeyer, 1996] Mayer, A. J. and Stockmeyer, L. J. (1996). The complexity of PDL with interleaving. *Theor. Comput. Sci.*, 161(1&2) :109–122.

[Peleg, 1987] Peleg, D. (1987). Concurrent dynamic logic. *J. ACM*, 34(2) :450–479.

[Rintanen et al., 2006] Rintanen, J., Heljanko, K., and Niemelä, I. (2006). Planning as satisfiability : parallel plans and algorithms for plan search. *Artif. Intell.*, 170(12-13) :1031–1080.

[Spalazzi and Traverso, 2000] Spalazzi, L. and Traverso, P. (2000). A dynamic logic for acting, sensing, and planning. *J. Log. Comput.*, 10(6) :787–821.

[Veloso et al., 2014] Veloso, P. A. S., Veloso, S. R. M., and Benevides, M. R. F. (2014). PDL for structured data : a graph-calculus approach. *Logic Journal of the IGPL*, 22(5) :737–757.

Knowledge Compilation for Action Languages

Sergej Scheck Alexandre Niveau Bruno Zanuttini
Normandie Univ.; UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France
sergej.scheck,alexandre.niveau,bruno.zanuttini@unicaen.fr

Résumé

Nous étudions différents langages permettant de représenter des actions non déterministes pour la planification automatique, du point de vue de la compilation de connaissances. Précisément, nous considérons la question de la concision des langages (quelle est la taille de la description d'une action dans chaque langage?) et des questions de complexité (quelle est la complexité algorithmique de décider si un état est un successeur d'un autre pour une action décrite dans l'un de ces langages?). Nous étudions une version abstraite et nondéterministe de PDDL, le langage des théories d'actions en NNF, et DL-PPA, la logique dynamique des affectations propositionnelles parallèles. Nous montrons que ces langages ont une concision différente, et une complexité de requête différente : DL-PPA est le plus concis et NNF le moins concis, et décider si un état est successeur d'un autre est déjà NP-complet pour PDDL nondéterministe.

Mots Clef

Planification, compilation de connaissances, logique dynamique des affectations propositionnelles, planning domain definition language, théories d'actions

Abstract

We study different languages for representing nondeterministic actions in automated planning from the point of view of knowledge compilation. Precisely, we consider succintness issues (how succinct is the description of an action in each language?) and complexity issues (how hard is it to decide whether a state is a successor of another one through some action described in one of these languages?). We study an abstract, nondeterministic version of PDDL, the language of NNF action theories, and DL-PPA, the dynamic logic of parallel propositional assignments. We show that these languages have different succinctness and different complexity of queries: DL-PPA is the most succinct one and NNF is the least succinct, and deciding successorship is already NP-complete for nondeterministic PDDL.

Keywords

Planning, knowledge compilation, dynamic logic of propositional assignments, planning domain definition language, action theories

1 Introduction

In automated planning, a central aspect of the description of problems is the formal representation of actions. Such representations are indeed needed for specifying the actions available to the agent (PDDL [15] is a standard language for this), and also for the planners to manipulate them while searching for a plan.

In this paper, we consider different representation languages from the point of view of knowledge compilation [9]. Knowledge compilation is the study of formal languages under the point of view of queries (how efficient is it to answer various queries depending on the language?), transformations (how efficient is it to transform or combine different representations in a given language?), and succinctness (how concise is it to represent knowledge in each language?). Most work in knowledge compilation has been done on representations of Boolean functions, for instance, by Boolean formulas in negation normal form, by ordered binary decision diagrams, etc. [9].

As far as we know there has been no systematic study of languages for representing actions per themselves. This is however an important problem, as planners need to query action representations again and again while searching for a plan (for instance, to find out which actions are applicable at the current node of the search tree), and typically start by transforming the action specifications into some representation suited for this. Hence having a clear picture of the properties of languages is clearly of interest for the development of such planners.

However, there have been a few papers studying aspects related to knowledge compilation for planning. For instance, Nebel has considered questions very similar to ours [17]. His study uses a rather powerful notion of compilation, where translations from one formal language to another are allowed to change the set of variables and the set of actions.[1] This captures compilation schemes where one is interested in preserving the existence of plans and their size. Contrastingly, we are interested in a strict notion of compilation, where the set of variables and the specification of initial states and goals are unchanged by the translation, while each action is translated into one with the same semantics. This is more demanding, but makes translations applicable in broader settings (for instance, to problems

[1] Actions are called "operators" there.

where we want to count or enumerate plans). Bäckström and Jonsson have studied representations of plans with respect to their size and to the complexity of retrieving the individual actions which they prescribe at each step [3]. This is also related to our work, but with a focus on languages for representing plans, while we study languages for representing actions.

We are interested here in (purely) nondeterministic actions, which lie at the core of fully observable nondeterministic planning and of conformant planning [19, 1, 12, 16, 20, 13]. We moreover consider propositional domains, in which states are assignments to a given set of propositions. The languages which we consider are of different natures: (grounded) **PDDL** is a specification language, **NNF** action theories are typically used as an internal representation by solvers, and **DL-PPA** is a logic allowing to specify programs and to reason about them. However, all of them can be viewed as languages for representing actions (as nondeterministic mappings from states to states), and their diversity (allowed constructs, representation of persisting values) allows us to give a clear picture. Our mid-term goal is to give a systematic picture of languages arising from all combinations of allowed constructs among the ones introduced in the literature (like nondeterministic choice, iteration, persistency by default, etc.).

The paper is structured as follows. In Section 2 we give the necessary background about actions and logic, and in Section 3 we formally define the action languages which we will consider. We then give our results: in Section 4 we prove positive results about polynomial-time translations between the languages, then in Section 5 we study the complexity of deciding whether a state is a possible successor of another state given an action description in one of these languages, and in Section 6 we give negative results about polynomial translations, which allows us to determine which languges are strictly more succinct than others. Finally, we conclude in Section 7.

2 Preliminaries

For any planning problem we consider a fixed finite set $P = \{p_1, \ldots, p_n\}$ of propositional variables. A subset of P is called a *P-state*, or simply a *state*. The intended interpretation of a state $s \in 2^P$ is the assignment to P in which all variables in s are true, and all variables in $P \setminus s$ are false. As an example, for $P = \{p_1, p_2, p_3\}$, $s = \{p_1, p_3\}$ denotes the state in which p_1 and p_3 are true and p_2 is false We write $\mathbb{P}$ for $\{p_i \mid i \in \mathbb{N}\}$.

Actions In this article we consider (purely) nondeterministic actions, which map states to sets of states. This means that a single state may have several successors through the same action, in contrast with deterministic actions (which map states to states), and that no relative likelihood is encoded between the successors of a state, in contrast with stochastic actions (which map states to probability distributions over states).

Definition 1 (action)**.** *Let P be a finite set of propositional variables. A* nondeterministic P-action *is a mapping a from 2^P to $2^{(2^P)}$. The elements of $a(s)$ are called* a-successors *of s.*

In the literature, actions are often considered together with preconditions which have to be satisfied to allow the execution of the action. However, for the results in this paper it is not important whether we require the action preconditions to be written explicitly, so for simplicity we assume them to be implicit. This means that an action a is applicable to a state s if and only if there exists at least one a-successor state s' of s.

Example 2. *Consider the following hunting example. Let*

$$P = \{\text{rabbit\_in\_sight}, \text{ rabbit\_alive}, \text{ loaded\_rifle}\}.$$

The action shoot_rabbit *can be described as "if* rabbit_alive *then: if* loaded_rifle *and* rabbit_in_sight*, then not* loaded_rifle *and either* rabbit_alive *and not* rabbit_in_sight *or not* rabbit_alive *and* rabbit_in_sight*, otherwise state unchanged". The action is applicable only if the rabbit is alive (otherwise it is not sensible to shoot at him). In this case, if the hunter is ready to shoot (the rifle is loaded and he can see the rabbit), then he tries to shoot the rabbit (he might miss the rabbit who hears the shot and runs away, so the action is nondeterministic), and if he is not ready to shoot, then nothing happens.*
Let $s = \{\text{rabbit\_in\_sight}, \text{ rabbit\_alive}, \text{ loaded\_rifle}\}$ *be the state where all three variables are true. Then* $\text{shoot\_rabbit}(s)$ *is the set of states ("successors")* $\{s', s''\}$ *with* $s' = s \setminus \{\text{rabbit\_alive}, \text{loaded\_rifle}\} = \{\text{rabbit\_in\_sight}\}$ *and* $s'' = s \setminus \{\text{rabbit\_in\_sight}, \text{loaded\_rifle}\} = \{\text{rabbit\_alive}\}$.

In this article, we are interested in the properties of *representations* of actions in various langugages.

Definition 3 (action language)**.** *An* action language *is an ordered pair $\langle L, I\rangle$, where L is a set of* action descriptions *and I is an* interpretation function*. Action descriptions are ordered pairs $\langle \alpha, P\rangle$ where α is a formula and P is a finite subset of $\mathbb{P}$. The interpretation function I maps every action description $\langle \alpha, P\rangle \in L$ to a P-action $I(\alpha, P)$.*

Observe that P is *a priori* not related to the variables of α (this depends on the language). For instance, variables of P not mentioned in an **NPDDL** expression α are assumed to persist, and a formula may also use variables outside of P and even outside of $\mathbb{P}$ (called *auxiliary* variables), as in **NNFAT**.

If the language $\langle L, I\rangle$ and the set P are clear from the context (or we just consider them to be fixed), then we write $\alpha(s)$ instead of $I(\alpha, P)(s)$ for the set of all α-successors of s.

In this article, we are mostly interested in translations between languages.

Definition 4 (translation)**.** *Let $\langle L_1, I_1\rangle$ and $\langle L_2, I_2\rangle$ be two action languages. A function $f : L_1 \to L_2$ is a* (proper) translation *if $I_1(\alpha, P) = I_2(f(\alpha, P), P)$ holds for all $\langle \alpha, P\rangle \in L_1$.*

In words, this means that the L_1-action description $\langle \alpha, P\rangle$ and the L_2-formula $f(\alpha, P)$ describe the same P-action. Again, when P is clear from the context, we write $f(\alpha)$ for $f(\alpha, P)$.
The function f is called a *polynomial-time* translation if it can be computed in time polynomial in the size of α and P. It is called a *polynomial-size* translation if the size of $f(\alpha, P)$ is bounded by a fixed polynomial in the size of α together with the size of P. Clearly, a polynomial-time translation is necessarily also a polynomial-size one, but a polynomial-size translation may not be polynomial-time.

Logic A Boolean formula φ is said to be in *negation normal form* (**NNF** for short) if it is built up from literals using conjunctions and disjunctions, *i.e.*, if it is generated by the grammar

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where p ranges over $\mathbb{P}$. We also use the shorthand notation $\top$ for $p \vee \neg p$ and $\bot$ for $p \wedge \neg p$, for an arbitrary $p \in \mathbb{P}$. For such a formula φ, $\mathrm{V}(\varphi)$ denotes the set of variables occuring in φ.
The set of formulas in **NNF** is complete, that is, every Boolean function can be described by an **NNF** formula. It is important to note that a formula φ with $\mathrm{V}(\varphi) \subseteq P$ for some set of variables P can be regarded as a formula over P (and the truth value of the corresponding Boolean function does not depend on the variables in $P \setminus \mathrm{V}(\varphi)$). For a boolean formula φ over a set of variables P and a state $s \subseteq P$, we write $s \models \varphi$ if φ evaluates to $\top$ under the assignment s.

Notation As a general rule, we use variables $a, b, \ldots$ for actions, $\alpha, \beta, \ldots$ for action expressions (in some language), and $\varphi, \psi, \ldots$ for logical formulas. Since action descriptions are also formulas in some language, we reserve the term "expression" for action descriptions and the term "formula" for logical formulas occuring in them.

Representations In the whole article we assume the expressions and formulas of action languages to be "flat", *i.e.*, that the amount of memory space required to store the expression or formula is its number of symbols (without the parentheses). This is to be contrasted with representations of NNF formulas (in particular) in which isomorphic subformulas are assumed to be represented only once, with any superformula pointing to this shared representation, a representation widely used in the literature about knowledge compilation [9]. We however wish to highlight that all our results would go through if we assumed such "circuit" (or "DAG") representations for formulas (we leave the case of circuit representations of expressions for future work).

3 Action Languages

In this section we formally define the action languages which we study later.

Variants of PDDL The first language which we consider is the well-known planning domain description language (**PDDL**). This language is a standardized one used for specifying actions at the relational level, widely used as an input for planners, especially in the international planning competitions [15, 10, 11]. Since we are interested in nondeterministic actions, we consider a nondeterministic variant of **PDDL** inspired by **NPDDL** [6], and so as to abstract away from the precise syntax of the specification language, we consider an idealized version. Finally, we consider a grounded version of **PDDL**, namely, a propositional one. Still we use the name "**NPDDL**", since we use essentially the same constructs.
We first define the syntax of **NPDDL**.

Definition 5 (**NPDDL** action descriptions)**.** *An* **NPDDL** action description *is an ordered pair $\langle \alpha, P\rangle$, where α is an expression generated by the grammar*

$$\alpha ::= \varepsilon \mid p \mid \neg p \mid \alpha \,\&\, \alpha \mid \varphi \rhd \alpha \mid (\alpha \mid \alpha)$$

where p ranges over P and φ over Boolean formulas in **NNF** *over P.*

Intuitively,

- ε describes the action with no effect (the only successor of s is s itself),
- p (resp. $\neg p$) is the action which makes p true (resp. false),
- $\&$ denotes simultaneous execution (with no successor if the operands are inconsistent together),
- $\rhd$ denotes conditional execution,
- $\mid$ denotes nondeterministic choice,

and, importantly, variables not explicitly modified by the action are assumed to keep their value.
We insist that this syntax is an idealization of nondeterministic (grounded) **PDDL**; for instance, the action which we write $x \rhd \big(y|(\neg y \,\&\, z)\big)$ would be written

```
when x (oneof y (and (not y) z))
```

with the syntax of **NPDDL** [6].
Action descriptions in **NPDDL** are interpreted as actions as follows.

Definition 6 (semantics of **NPDDL**)**.** *The* interpretation function for **NPDDL** *is the function I defined by $I(\alpha, P)(s) = \{(s \cup e^+) \setminus e^- \mid \langle e^+, e^-\rangle \in M(\alpha, s)\}$, where $M(\alpha, s)$ is the set of possible modifications of s caused by α, defined inductively by*

- $M(\varepsilon, s) = \{\langle \emptyset, \emptyset\rangle\}$,

- $M(p,s) = \{\langle\{p\},\emptyset\rangle\}$ *and* $M(\neg p,s) = \{\langle\emptyset,\{p\}\rangle\}$,
- $M(\varphi \rhd \alpha, s) = M(\alpha,s)$ *if* $s \models \varphi$, *else* $\{\langle\emptyset,\emptyset\rangle\}$,
- $M(\alpha_1 \,\&\, \alpha_2, s) = \{\langle e_1^+ \cup e_2^+, e_1^- \cup e_2^-\rangle \mid \langle e_1^+, e_1^-\rangle \in M(\alpha_1,s), \langle e_2^+, e_2^-\rangle \in M(\alpha_2,s), e_1^+ \cap e_2^- = e_1^- \cap e_2^+ = \emptyset\}$,
- $M(\alpha_1 \mid \alpha_2, s) = M(\alpha_1,s) \cup M(\alpha_2,s)$.

When we want to denote simultaneous execution of all action descriptions in a set A, we write $\&_{\alpha\in A}\,\alpha$. Also note that the action description $p \,\&\, \neg p$ (for an arbitrary $p \in \mathbb{P}$) defines a nonexecutable action. Hence it can be used as a subaction for encoding a precondition, and we use $\bot$ as shorthand notation for it.

Example 7 (continued)**.** *The following is an* **NPDDL** *action description for the action* shoot_rabbit *of Example 2.*

$$\begin{aligned} & (\neg\text{rabbit\_alive} \rhd \bot) \\ \&\quad & \big((\text{rabbit\_alive} \wedge \text{rabbit\_in\_sight} \wedge \text{loaded\_rifle}) \\ & \quad \rhd (\neg\text{loaded\_rifle} \\ & \qquad \&(\neg\text{rabbit\_alive} \mid \neg\text{rabbit\_in\_sight}))\big) \end{aligned}$$

We are also interested in the language **NPDDL** as extended by the sequential execution operator ";".

Definition 8 (**NPDDL**$_{\text{seq}}$)**.** *The language* **NPDDL**$_{\text{seq}}$ *is the language in which action descriptions are generated by the following grammar:*

$$\alpha ::= \varepsilon \mid p \mid \neg p \mid \alpha \,\&\, \alpha \mid \varphi \rhd \alpha \mid (\alpha \mid \alpha) \mid \alpha;\alpha$$

and the interpretation function is the same as that of **NPDDL** *for all constructs, augmented with*

$$(\alpha_1;\alpha_2)(s) = \{s'' \mid \exists s' \in \alpha_1(s) : s'' \in \alpha_2(s')\}$$

NNF action theories We now define the second language which we consider, namely that of (**NNF**) action theories. Such representations are typically used by planners which reason explicitly on *sets of states* (aka *belief states*), since they allow for symbolic operations on belief states and action descriptions [8, 7, 20]. We consider action theories represented in **NNF**, which encompasses representations usually used like OBDDs or DNFs.
To prepare the definition we associate a variable $p' \in \mathbb{P}'$ to each variable $p \in \mathbb{P}$, where $\mathbb{P}'$ is a disjoint copy of $\mathbb{P}$; p' is intended to denote the value of p *after* the action took place, while p denotes the value *before*.

Definition 9 (**NNFAT**)**.** *An* **NNFAT** *action description is an ordered pair* $\langle\alpha, P\rangle$ *where* α *is a Boolean formula in* **NNF** *over the set of variables* $P \cup \{p' \mid p \in P\}$. *The interpretation of* $\langle\alpha, P\rangle$ *is defined by*

$$I(\alpha,P)(s) = \{s' \mid s \cup \{p' \mid p \in s'\} \models \alpha\}$$

In words, an (**NNF**) action theory represents the set of all ordered pairs $\langle s, s'\rangle$ such that s' is a successor of s, as a Boolean formula over variables in $P \cup \{p' \mid p \in P\}$.
Importantly, **NNFAT** does not assume the frame axiom, so that if, for example, a variable does not appear at all in an **NNFAT** action description, then this means that its value after the execution of the action can be arbitrary. For instance, the action description $\langle x' \vee (\neg y \vee z'), \{x,y,z\}\rangle$ represents an action which either (1) sets x to true and y, z to any value (nondeterministically), or (2) sets z to true and x, y to any value, in case y is true in the initial state, and otherwise sets each variable to any value, or (3) performs any consistent combination of (1) and (2).
Observe that a conjunct over variables in P in an **NNFAT** action description in fact encodes a precondition.

Example 10 (continued)**.** *The action* shoot_rabbit *of Example 2 can be written as (we use* $\rightarrow$ *and* $\leftrightarrow$ *for readability)*

$$\begin{aligned} & \text{rabbit\_alive} \\ \wedge\quad & (\text{loaded\_rifle} \wedge \text{rabbit\_in\_sight}) \\ & \rightarrow \Big(\big((\neg\text{rabbit\_in\_sight}' \wedge \text{rabbit\_alive}') \\ & \qquad \vee (\text{rabbit\_in\_sight}' \wedge \neg\text{rabbit\_alive}')\big) \\ & \qquad \wedge \big(\neg\text{loaded\_rifle}'\big)\Big) \\ \wedge\quad & \big((\text{rabbit\_alive} \not\leftrightarrow \neg\text{rabbit\_alive}') \\ & \quad \vee (\text{rabbit\_in\_sight} \not\leftrightarrow \text{rabbit\_in\_sight}') \\ & \quad \vee (\text{loaded\_rifle} \not\leftrightarrow \text{loaded\_rifle}')\big) \\ & \rightarrow (\text{loaded\_rifle} \wedge \text{rabbit\_in\_sight}) \end{aligned}$$

As can be seen, encoding in **NNFAT** *the fact that the values of variables persist unless stated otherwise, typically requires subformulas (here the last conjunct) playing the same role as successor-state axioms in the situation calculus [18]. This typically requires a lot of space. We will give a formal meaning to this remark later in the paper (Proposition 30).*

Obviously, every action can be represented in this language, since the language of **NNF** formulas is complete for Boolean functions. We will see later that all the other languages that we study in this article are at least as succinct as **NNFAT**; hence in particular, they are all complete as well.

DL-PPA The last language that we consider in this paper is the *dynamic logic of parallel propositional assignments* (**DL-PPA** for short), which has been introduced by Herzig *et al.* as an extension of the language **DL-PA** [14]. **DL-PA** was initially proposed for reasoning about imperative programs [5]. For instance, deciding whether there exists a plan from a given initial state to a goal characterized by a Boolean formula φ using actions $\alpha_1, \ldots, \alpha_k$ amounts to deciding whether the initial state satisfies the **DL-PA** formula $\langle(\alpha_1 \cup \ldots \cup \alpha_k)^*\rangle\varphi$. However, **DL-PPA** can also be used as an action language [14].

Definition 11 (**DL-PPA** action descriptions)**.** *A* **DL-PPA** action description *is an ordered pair* $\langle\alpha, P\rangle$

where α is an expression generated by the following grammar:

$$\begin{array}{lll} \alpha & ::= & p \leftarrow \varphi \mid \varphi? \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha \sqcap \alpha \mid \alpha \sqcup \alpha \mid \ \alpha^* \\ \varphi & ::= & p \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\alpha\rangle\varphi \end{array}$$

where p ranges over P.

In the literature, action descriptions are typically called **DL-PPA** *programs*, and formulas φ as in the definition are typically called **DL-PPA** *formulas*. Intuitively, the symbols mean the following:

- $p \leftarrow \varphi$ evaluates φ in the current state and assigns the resulting value to p,
- $\varphi?$ tests whether φ is satisfied in the current state and fails if it is not the case,
- ; denotes sequential execution,
- $\cup$ denotes (exclusive) nondeterministic choice, that is, execution of exactly one subaction,
- $\sqcap$ denotes parallel execution,
- $\sqcup$ denotes nonexclusive nondeterministic choice, that is, execution of one subaction or of both subactions,
- $*$ denotes looping an arbitrary number of times,
- $\langle\alpha\rangle\varphi$ denotes the modal construction "there is an execution of α ending in a state which satisfies φ".

We also use the shorthand notation $+p$ (resp. $-p$) for $p \leftarrow \top$ (resp. $p \leftarrow \bot$) and $[\alpha]\varphi$ for $\neg\langle\alpha\rangle\neg\varphi$ ("all executions of α end in a state which satisfies φ"). Moreover, as follows from the semantics which is defined below, $\top?$ denotes an empty action mapping any state to itself, and $(\varphi?;\alpha) \cup (\neg\varphi?;\beta)$ denotes the construction "if φ then α else β".

Also observe that every **NNF** formula can be rewritten into an equivalent **DL-PPA**-formula (without $\wedge$) in linear time, since using De Morgan's laws we can always rewrite $\varphi \wedge \psi$ into the logically equivalent formula $\neg(\neg\varphi \vee \neg\psi)$.

Suppose that we are given the set P. To every formula there is an associated valuation which is the set of states that are models of the formula. The interpretations of programs are ternary relations on the set of states 2^P. We denote the valuation of a formula and the interpretation of a program by $\|\varphi\|$ and $\|\alpha\|$ respectively. $(s, s', w) \in \|\alpha\|$ means that there is an execution of α that leads from s to s' by assigning the variables in w. The formulas in **DL-PPA** are interpreted as follows, where $s, s', \hat{s} \ldots$ denote states and $w, w_1, \hat{w}_1 \ldots$ denote subsets of P:

Definition 12.

$$\begin{aligned} \|p\| &= \{s \mid p \in s\} \\ \|\top\| &= 2^P \\ \|\neg\varphi\| &= 2^P \setminus \|\varphi\| \\ \|\varphi_1 \vee \varphi_2\| &= \|\varphi_1\| \cup \|\varphi_2\| \\ \|\langle\alpha\rangle\varphi\| &= \{s \mid \exists w\ \exists s' : (s, s', w) \in \|\alpha\| \wedge s' \in \|\varphi\|\} \end{aligned}$$

The programs are interpreted in the following way:

$$\begin{aligned} \|p \leftarrow \varphi\| =& \{(s, s \cup \{p\}, \{p\}) \mid s \in \|\varphi\|\} \\ & \cup \{(s, s \setminus \{p\}, \{p\}) \mid s \notin \|\varphi\|\} \\ \|\varphi?\| =& \{(s, s, \emptyset) \mid s \in \|\varphi\|\} \\ \|\alpha_1;\alpha_2\| =& \{(s, s', w) \mid \exists \hat{w}_1\ \exists \hat{w}_2\ \exists \hat{s}' : (s, \hat{s}, \hat{w}_1) \in \|\alpha_1\| \\ & \wedge\ (\hat{s}, s', \hat{w}_2) \in \|\alpha_2\| \ \wedge\ w = \hat{w}_1 \cup \hat{w}_2\} \\ \|\alpha_1 \cup \alpha_2\| =& \|\alpha_1\| \cup \|\alpha_2\| \\ \|\alpha_1 \sqcap \alpha_2\| =& \{(s, s', w) \mid \exists s'_1\ \exists s'_2\ \exists w_1\ \exists w_2 : (s, s'_1, w_1) \\ & \in \|\alpha_1\|)\} \wedge (s, s'_2, w_2) \in \|\alpha_2\| \wedge w_1 \cap w_2 \cap s'_1 \\ & = w_1 \cap w_2 \cap s'_2 \wedge w = w_1 \cup w_2 \\ & \wedge s' = (s \setminus w) \cup (s'_1 \cap w_1) \cup (s'_2 \cap w_2) \\ \|\alpha_1 \sqcup \alpha_2\| =& \|\alpha_1 \cup \alpha_2 \cup (\alpha_1 \sqcap \alpha_2)\| \\ \|\alpha^*\| =& \bigcup_{k \in \mathbb{N}} \|\underbrace{\alpha;\alpha;\ldots;\alpha}_{k \text{ times}}\| \end{aligned}$$

When applying **DL-PPA** *to planning tasks we identify valuations with states and describe actions as programs: an action α is described by a program α with $\alpha(s) = \{s' \mid \exists w : (s, w, s') \in \|\alpha\|\}$.*

Finally, we will be interested in the restriction of **DL-PPA** obtained when disallowing nonexclusive choice, the Kleene star, and modalities.

Definition 13 (restricted **DL-PPA**)**.** *The language* restricted **DL-PPA** *is the language in which action descriptions $\langle\alpha, P\rangle$ are generated by the following grammar:*

$$\begin{array}{lll} \alpha & ::= & p \leftarrow \varphi \mid \varphi? \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha \sqcap \alpha \\ \varphi & ::= & p \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \end{array}$$

where p ranges over P, and whose semantics is the same as **DL-PPA** *restricted to this language.*

Example 14 (continued)**.** *The action* shoot_rabbit *of our running example 2 can be described as follows in (restricted)* **DL-PPA***:*

$$\begin{array}{l} \text{rabbit\_alive?;} \\ \big((\text{rabbit\_in\_sight} \wedge \text{loaded\_rifle})?; \\ \quad (-\text{rabbit\_alive} \cup -\text{rabbit\_in\_sight}); \\ \quad -\text{loaded\_rifle}\big) \\ \cup\big(\neg\text{rabbit\_in\_sight} \vee \neg\text{loaded\_rifle?}\big) \end{array}$$

Example 15. *The following* **DL-PPA** *program illustrates the meaning of the modal operators and of the Kleene star:*

$$\begin{array}{l} \big(\langle\text{shoot\_rabbit};\text{shoot\_rabbit}^*\rangle\neg\text{rabbit\_alive?}; \\ \quad \text{shoot\_rabbit}\big) \\ \cup\big(-\text{loaded\_rifle}\big) \end{array}$$

This action can be read as follows. If the hunter has a chance to kill the rabbit (which especially means that in the current state the rabbit is alive, as ensured by the first occurrence of shoot_rabbit*), then the hunter will shoot. Otherwise he will be disappointed and shoot in the air because he has no hope for success. But there could be several reasons for him being unable to kill the rabit: the rabbit is already dead, or the rifle is not loaded, or he does not see the rabbit...*

Note that **DL-PPA** has all the features of **NPDDL**, like the implicit frame axiom, and it additionally allows for modal operators. Hence, summarizing, we study languages with and without the sequence operator, with and without the implicit frame axiom, and with and without modalities. A mid-term goal of our work is to study combinations of such features in a systematic way, and we view this restricted set of languages as a meaningful set of representative languages to start with.

4 Polynomial-Time Translations

In this section, we exhibit translations between some of our languages of interest which can be carried out in polynomial time (hence, *a fortiori*, are polynomial-size). We remark that the identity function is an obvious polynomial-time translation from restricted **DL-PPA** into **DL-PPA**. We first show that any **NNFAT** action description α can be translated in polynomial time to an **NPDDL** action description $f(\alpha)$. The translation looks like a simple rewriting of α, but we have to care about (1) the fact that in **NNFAT**, a variable not explicitly set to a value can take any value in the next state s', contrary to persistency by default in **NPDDL**, and (2) the fact that $\vee$ is inclusive-or in **NNFAT**, while nondeterministic choice in **NPDDL** is interpreted as one effect taking place (but not both). For (1) we will make explicit in the **NPDDL** translation that these variables can take any value, and for (2) it will turn out that in the translation of $\alpha_1 \vee \alpha_2$ into $f(\alpha_1) \,|\, f(\alpha_2)$, $f(\alpha_1)$ will encode all possible transitions of α_1, including those of $\alpha_1 \wedge \alpha_2$ (the "inclusive part" of the $\vee$), and similarly for $f(\alpha_2)$.
The translation f is defined inductively as follows for an **NNFAT** action description $\langle \alpha, P \rangle$:

1. if $\mathrm{V}(\alpha) \subseteq P$, then $f(\alpha) = \big(\neg\alpha \rhd \bot\big) \,\&\, \big(\alpha \rhd (\mathop{\&}_{p\in P}(p \,|\, \neg p))\big)$;
2. if $\mathrm{V}(\alpha) \not\subseteq P$ and $\mathrm{V}(\alpha_1) \subseteq P$, then $f(\alpha_1 \vee \alpha_2) = \big(\neg\alpha_1 \rhd f(\alpha_2)\big) \,\&\, \big(\alpha_1 \rhd (\mathop{\&}_{p\in P}(p \,|\, \neg p))\big)$; dually for $\mathrm{V}(\alpha_2) \subseteq P$;
3. if $\mathrm{V}(\alpha) \not\subseteq P$ and $\mathrm{V}(\alpha_1) \subseteq P$, then $f(\alpha_1 \wedge \alpha_2) = \big(\alpha_1 \rhd f(\alpha_2)\big) \,\&\, \big(\neg\alpha_1 \rhd \bot\big)$; dually for $\mathrm{V}(\alpha_2) \subseteq P$;
4. $f(p') = p \,\&\, \Big(\mathop{\&}_{q\in P, q\neq p}(q \,|\, \neg q)\Big)$;
5. $f(\neg p') = \neg p \,\&\, \Big(\mathop{\&}_{q\in P, q\neq p}(q \,|\, \neg q)\Big)$;
6. if $\mathrm{V}(\alpha_1), \mathrm{V}(\alpha_2) \not\subseteq P$, $f(\alpha_1 \wedge \alpha_2) = f(\alpha_1) \,\&\, f(\alpha_2)$;
7. if $\mathrm{V}(\alpha_1), \mathrm{V}(\alpha_2) \not\subseteq P$, $f(\alpha_1 \vee \alpha_2) = f(\alpha_1) \,|\, f(\alpha_2)$.

Observe for future reference that for all states s, all possible modifications $\langle e^+, e^- \rangle$ in $M(f(\alpha), s)$ are P-complete in the sense that $e^+ \cup e^- = P$ (all variables are mentioned in e^+ or e^-). This is easily seen by induction on the definition of f.

Proposition 16. *Let $\langle \alpha, P \rangle$ be an* **NNFAT** *action description. Then we have $s' \in \alpha(s) \iff s' \in f(\alpha)(s)$.*

PROOF. The translation is clearly polynomial-time, since the "gadgets" added to the rewriting in the first 5 cases involve no recursive call of f. We now show that it is correct, by induction on the structure of α.

1. First assume $\mathrm{V}(\alpha) \subseteq P$. Then by the semantics of **NNFAT**, $s' \in \alpha(s)$ holds if and only if s satisfies α, which is equivalent to s satisfying α and s' being arbitrary, which is equivalent to $s' \in f(\alpha)(s)$ by the definition of $f(\alpha)$ and the semantics of **NPDDL**.
2. Now assume $\alpha = \alpha_1 \vee \alpha_2$ and $\mathrm{V}(\alpha_1) \subseteq P$. For $s' \in \alpha_1(s)$, we have $s' \in \alpha(s)$, and since we have $\mathrm{V}(\alpha_1) \subseteq P$, we have that s satisfies α_1, and hence $s' \in f(\alpha)(s)$ is equivalent to $s' \in \big(\mathop{\&}_{p\in P}(p \,|\, \neg p)\big)(s)$, which is true for all s'; hence both $s' \in \alpha(s)$ and $s' \in f(\alpha)(s)$ hold. Now for $s' \notin \alpha_1(s)$, we have $s' \in \alpha(s)$ if and only if $s' \in \alpha_2(s)$, which is equivalent to $s' \in f(\alpha_2)(s)$ by the induction hypothesis, and this in turn is equivalent to $s' \in f(\alpha)(s)$ by the definition of $f(\alpha_1 \vee \alpha_2)$ and the semantics of **NPDDL**.
3. Now assume $\alpha = \alpha_1 \wedge \alpha_2$ and $\mathrm{V}(\alpha_1) \subseteq P$. We have that $s \in \alpha(s)$ is equivalent to $s' \in \alpha_1(s) \wedge s' \in \alpha_2(s)$, and since we have $\mathrm{V}(\alpha_1) \subseteq P$, this is equivalent to $s \models \alpha_1 \wedge s' \in \alpha_2(s)$, which in turn is equivalent to $s \models \alpha_1 \wedge s' \in f(\alpha_2)(s)$ by the induction hypothesis, which is finally equivalent to $s' \in f(\alpha)(s)$ by the definition of $f(\alpha_1 \wedge \alpha_2)$ and the semantics of **NPDDL**.
4. Now let $\alpha = p'$. Then $s' \in \alpha(s)$ is equivalent to $p \in s'$ with s' otherwise arbitrary, which is clearly equivalent to $s' \in f(\alpha)(s)$.
5. The proof for $\alpha = \neg p'$ is symmetric to the previous case.
6. Let $\alpha = \alpha_1 \wedge \alpha_2$. Then $s' \in \alpha(s)$ is equivalent to $s' \in \alpha_1(s) \wedge s' \in \alpha_2(s)$, and by the induction hypothesis this is equivalent to $s' \in f(\alpha_1)(s) \wedge s' \in f(\alpha_2)(s)$. Now since the possible modifications of $f(\alpha_1)$ and $f(\alpha_2)$ are P-complete, it is easily seen from the definition of the semantics of **NPDDL** that the

set of possible modifications $M(f(\alpha_1) \,\&\, f(\alpha_2), s)$ is exactly $M(f(\alpha_1), s) \cap M(f(\alpha_2), s)$, so that $s' \in f(\alpha_1)(s) \wedge s' \in f(\alpha_2)(s)$ is equivalent to $s' \in \big(f(\alpha_1) \,\&\, f(\alpha_2)\big)(s)$, that is, to $s' \in f(\alpha)(s)$.

7. Finally let $\alpha = \alpha_1 \vee \alpha_2$. Assume first $s' \in \alpha(s)$, and by symmetry $s' \in \alpha_1(s)$; then by the induction hypothesis we have $s' \in f(\alpha_1)(s)$ and hence, $s' \in \big(f(\alpha_1) \,|\, f(\alpha_2)\big)(s) = f(\alpha)(s)$. Conversely, assume $s' \in f(\alpha)(s)$, then by the definition of $f(\alpha)$ and the semantics of $|$ we have $s' \in f(\alpha_1)(s)$ or $s' \in f(\alpha_2)(s)$. Assume by symmetry $s' \in f(\alpha_1)(s)$. Then by the induction hypothesis we have $s' \in \alpha_1(s)$ and hence, $s' \in (\alpha_1 \vee \alpha_2)(s)$, that is, $s' \in \alpha(s)$.

□

The following propositions are quite intuitive, because $\mathbf{NPDDL}_{\text{seq}}$ and restricted **DL-PPA** are essentially the same: $\varphi \rhd \ldots$ is analogous to $\varphi?$, $|$ is analogous to $\cup$, and $\&$ is analogous to $\sqcap$. However, we must pay attention to two facts. The first difference between the languages is that in $\mathbf{NPDDL}_{\text{seq}}$, if φ is not true in $\varphi \rhd \alpha$, then the action just does not change the current state whereas in **DL-PPA**, φ being false results in a failure. The other difference is that formulas in $\mathbf{NPDDL}_{\text{seq}}$ must be in **NNF**, while **DL-PPA** does not have restrictions on the occurence of $\neg$ but does not have the $\wedge$ connective.

Proposition 17. *There is a polynomial-time translation of* $\mathbf{NPDDL}_{\text{seq}}$ *into restricted* **DL-PPA**.

PROOF. Consider an $\mathbf{NPDDL}_{\text{seq}}$ action description α. The translation f first replaces each subformula of the form $\varphi \wedge \psi$ in α with $\neg(\neg\varphi \vee \neg\psi)$, then it computes an action description in restricted **DL-PPA** as follows:

$$\begin{array}{lcl} f(\epsilon) & = & \top? \\ f(p) & = & +p \\ f(\neg p) & = & -p \\ f(\alpha_1 \,\&\, \alpha_2) & = & f(\alpha_1) \sqcap f(\alpha_2) \\ f(\varphi \rhd \alpha) & = & \big(\varphi?; f(\alpha)\big) \cup (\neg\varphi?) \\ f(\alpha_1 \,|\, \alpha_2) & = & f(\alpha_1) \cup f(\alpha_2) \\ f(\alpha_1; \alpha_2) & = & f(\alpha_1); f(\alpha_2) \end{array}$$

It is easy to check that this translation can be computed in polynomial time and that it is correct. In particular, φ is duplicated in the fifth line but it involves no recursive call of f, hence preserving polynomial size (the rewriting of $\neg\varphi$ into a **DL-PPA** formula can be done in linear time), and $\neg\varphi?$ in the same line ensures that the action does nothing but does not fail when φ is not satisfied. □

The proof of the converse is completely symmetric.

Proposition 18. *There is a polynomial-time translation of restricted* **DL-PPA** *to* $\mathbf{NPDDL}_{\text{seq}}$.

PROOF. Consider a restricted **DL-PPA** action description $\langle \alpha, P \rangle$. The translation f first replaces each subformula of the form $\neg(\varphi \vee \psi)$ with $(\neg\varphi \wedge \neg\psi)$, ending up with a description in which all formulas are in **NNF**, then it computes an action description in $\mathbf{NPDDL}_{\text{seq}}$ as follows:

$$\begin{array}{lcl} f(p \leftarrow \varphi) & = & (\varphi \rhd p) \,\&\, (\neg\varphi \rhd \neg p) \\ f(\varphi?) & = & (\varphi \rhd \varepsilon) \,|\, (\neg\varphi \rhd \bot) \\ f(\alpha_1; \alpha_2) & = & f(\alpha_1); f(\alpha_2) \\ f(\alpha_1 \cup \alpha_2) & = & f(\alpha_1) \,|\, f(\alpha_2) \\ f(\alpha_1 \sqcap \alpha_2) & = & f(\alpha_1) \,\&\, f(\alpha_2) \end{array}$$

It is easy to check that this translation can be computed in polynomial time and that it is correct. In particular, φ is duplicated in the first and second lines but it involves no recursive call of f, hence preserving polynomial size, and $\neg\varphi \rhd \bot$ in the second line ensures that the action fails when φ is not satisfied. □

5 Complexity of Deciding Successorship

We now turn to studying the complexity of *queries* to action descriptions. In this paper, we concentrate on the most natural query, which is formally defined by the following computational problem.

Definition 19 (IS-SUCC)**.** *Let L be an action language. The decision problem* IS-SUCC *is defined by:*

- *input: an action description $\langle \alpha, P \rangle \in L$ and two states $s, s' \subseteq P$,*
- *question: is s' an α-successor of s?*

Proposition 20. *The problem* IS-SUCC *is polynomial-time solvable for $L =$* **NNFAT**.

PROOF. From the semantics of **NNFAT** it follows that deciding $s' \in \alpha(s)$ amounts to deciding whether the assignment to $P \cup \{p' \mid p \in P\}$ induced by s, s' satisfies α, which can clearly be done in linear time. □

Proposition 21. *The problem* IS-SUCC *is in* **NP** *for $L =$* $\mathbf{NPDDL}_{\text{seq}}$.

PROOF. We define a witness for a positive instance to be composed of either α_1 or α_2 for each subexpression $\alpha_1 \,|\, \alpha_2$ of α, and of a state t for each subexpression $\alpha_1; \alpha_2$ (representing the guessed intermediate state of the execution). Such a witness is clearly of polynomial size. Now verifying it amounts to verifying that when the nondeterministic choices are those encoded by the witness and the execution of sequence constructs go through the encoded intermediate states, s' is indeed an α-successor of s. This can clearly be done in polynomial time since there remains only to evaluate conditions of $\rhd$ constructs in given states and applying effects of the form p or $\neg p$. □

For showing hardness, we build a specific action able to "produce" all and only satisfiable 3-CNF formulas. For this we first define an encoding of any 3-CNF formula φ over n variables as an assignment to a polynomial number of variables.

Notation 22. *Let $n \in \mathbb{N}$ and X_n be the set of variables $\{x_1, \ldots, x_n\}$. Observe that there are a cubic number N_n of clauses of length 3 over X_n (any choice of 3 variables with a polarity for each). We fix an arbitrary enumeration $\gamma_1, \gamma_2, \ldots, \gamma_{N_n}$ of all these clauses, and we define Q_n to be the set of variables $\{q_1, q_2, \ldots, q_{N_n}\}$. Then we identify an assignment s to Q_n to the* 3-CNF *formula over X_n, written $\varphi(s)$, which for all i contains the clause γ_i if and only if $q_i \in s$ holds.*
We also write $s(\varphi)$ for the assignment to Q_n which encodes a 3-CNF *formula φ over X_n. By $\ell \in \gamma_i$ we mean that the literal ℓ occurs in the clause γ_i.*

Example 23. *Let $n = 2$, and consider an enumeration of all clauses over variables $X_2 = \{x_1, x_2\}$ which starts with $\gamma_1 = (x_1 \vee x_1 \vee x_2), \gamma_2 = (x_1 \vee x_1 \vee \neg x_2), \gamma_3 = (x_1 \vee \neg x_1 \vee x_2), \gamma_4 = (x_1 \vee \neg x_1 \vee \neg x_2), \gamma_5 = (\neg x_1 \vee \neg x_1 \vee x_2), \gamma_6 = (\neg x_1 \vee \neg x_1 \vee \neg x_2), \ldots$. Then the* 3-CNF *$\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_1 \vee x_2)$ is identified to the state $s(\varphi) = \{q_1, q_5\}$.*

Using Notation 22, for all $n \in \mathbb{N}$ we define the **NPDDL** action description $\langle \beta_n, Q_n \rangle$ by

$$\beta_n = \mathop{\&}_{x \in X_n} \left(\left(\mathop{\&}_{\gamma_i : x \in \gamma_i} (q_i \mid \varepsilon) \right) \mid \left(\mathop{\&}_{\gamma_i : \neg x \in \gamma_i} (q_i \mid \varepsilon) \right) \right)$$

Intuitively, β_n chooses an assignment ($\bot$ or $\top$) to each variable in X_n (outermost nondeterministic choices). Whenever it chooses one, it chooses nondeterministically some clauses which are satisfied by it, and adds them to the result. Hence it builds a satisfiable formula (which is satisfied precisely by—at least—the assignment made of its choices over each variable).

Lemma 24. *Let $n \in \mathbb{N}$ and let φ be a* 3-CNF *formula over X_n. Then φ is satisfiable if and only if $s(\varphi)$ is a β_n-successor of the state $\emptyset$.*

PROOF. If φ is satisfiable, let s_X be an assignment to X_n which satisfies it. Consider the execution of β_n in which for each x, when the subexpression corresponding to x is executed, the left (resp. right) subexpression of the nondeterministic choice is executed if $x \in s_X$ holds (resp. if $x \notin s_X$ holds). Finally, consider the execution of this expression $\&(q_i \mid \varepsilon)$ in which for all i, q_i (resp. ε) is executed when φ contains (resp. does not contain) the clause γ_i. Clearly, this execution reaches $s(\varphi)$. Conversely, if an execution reaches a state s, then a model of $\varphi(s)$ can be built by considering each variable $x \in X_n$, and including (resp. not including) x if and only if the execution went through the left (resp. right) of the nondeterministic choice when the subexpression corresponding to x was executed. □

Since β_n can clearly be built in polynomial time given a set of variables X_n, Lemma 24 directly gives a reduction from the 3-SAT problem to the problem IS-SUCC for **NPDDL**. Hence the latter problem is **NP**-hard, and since we have shown IS-SUCC to be in **NP** for $\mathbf{NPDDL}_{\mathrm{seq}}$ (Proposition 21), we have the following.

Proposition 25. *The problem* IS-SUCC *is* **NP***-complete for $L = \mathbf{NPDDL}$ and for $L = \mathbf{NPDDL}_{\mathrm{seq}}$.*

Finally, since $\mathbf{NPDDL}_{\mathrm{seq}}$ and restricted **DL-PPA** are translatable into each other in polynomial time (Propositions 17 and 18), we have the following.

Corollary 26. *The problem* IS-SUCC *is* **NP***-complete when L is restricted* **DL-PPA**.

We finally turn to the complexity of IS-SUCC for **DL-PPA**.

Proposition 27. *The problem* IS-SUCC *is* **PSPACE***-complete for $L =$* **DL-PPA**.

PROOF. It is known that model checking for **DL-PPA** is **PSPACE**-complete [4]. This problem is the one of checking whether a given state s is in $\|\varphi\|$ for a given **DL-PPA** formula φ. We reduce it to IS-SUCC for **DL-PPA** as follows.
Suppose that we are given a **DL-PPA** formula φ over the set $P = \{p_1, \ldots, p_n\}$, and let without loss of generality $s = \{p_1, \ldots, p_k\}$. Let r (standing for "result") be a fresh variable, and build the **DL-PPA** action description $\langle \alpha, P \cup \{r\} \rangle$ with

$$\begin{aligned} \alpha \quad = \quad & (\langle +p_1; \ldots; +p_k; -p_{k+1}; \ldots; -p_n \rangle \varphi?; +r) \\ & \cup (\neg \langle +p_1; \ldots; +p_k; -p_{k+1}; \ldots; -p_n \rangle \varphi?; -r) \end{aligned}$$

Clearly, α can be built in time polynomial in the size of φ, and α does nothing except setting r to $\top$ if $s \in \|\varphi\|$ holds, and to $\bot$ otherwise. It follows that s is a model of φ if and only if the state $\{r\}$ is an α-successor of the state $\emptyset$.[2]
Hence IS-SUCC is **PSPACE**-hard for $L =$ **DL-PPA**. For membership, we reduce it to the satisfiability problem for **DL-PPA** formulas, which is in **PSPACE** [4]. Given an action description $\langle \alpha, P \rangle$ with $P = \{p_1, \ldots, p_n\}$, a state $s = \{p_1, \ldots, p_k\}$ (without loss of generality) and a state s', we define φ to be the **DL-PPA** formula

$$\langle +p_1; \ldots; +p_k; -p_{k+1}; \ldots; -p_n; \alpha \rangle \Big(\bigwedge_{p \in s'} p \wedge \bigwedge_{p \in P \setminus s'} \neg p \Big)$$

Clearly, φ can be built in polynomial time, and it is satisfiable if and only if the program "go to state s and then execute α" can lead to the state s', which is just a rephrasing of s' being an α-successor of s. □

[2] The choice of $\emptyset$ is arbitrary, since the variables of φ are all set by the modalities and are used only there and hence, their initial and final values do not matter.

6 Succinctness

In this section, we study the relative succinctness of action languages. Succinctness is formally defined as follows [9].

Definition 28 (succinctness)**.** *An action language L_1 is said to be* at least as succinct as *an action language L_2, denoted by $L_1 \preceq L_2$, if there exists a polynomial-size translation from L_2 to L_1. If $L_1 \preceq L_2$ and $L_2 \not\preceq L_1$ hold, then L_1 is said to be* strictly more succinct than *L_2, written $L_1 \prec L_2$. If $L_1 \preceq L_2$ and $L_2 \preceq L_1$ hold, then L_1 and L_2 are said to be* equally succinct.

The succinctness relation $\preceq$ is reflexive and transitive, hence it is a preorder. However, it is not antisymmetric and thus not an order.
Clearly, if there is a polynomial-time translation from L_2 to L_1 then $L_1 \preceq L_2$ holds. Hence we have the following as a direct consequence of Propositions 17 and 18.

Proposition 29. *The languages $\mathbf{NPDDL}_{\mathrm{seq}}$ and restricted $\mathbf{DL\text{-}PPA}$ are equally succinct.*

Our next results rely on assumptions about *nonuniform* complexity classes. Recall that $\mathbf{P/poly}$ (resp. $\mathbf{NP/poly}$) is the class of all decision problems such that for all $n \in \mathbb{N}$, there is a polynomial-time algorithm (resp. a nondeterministic polynomial-time algorithm) which decides the problem for all inputs of size n [2]. The assumptions $\mathbf{NP} \not\subseteq \mathbf{P/poly}$ and $\mathbf{PSPACE} \not\subseteq \mathbf{NP/poly}$ which we use are standard ones; in particular, $\mathbf{NP} \subseteq \mathbf{P/poly}$ would imply a collapse of the polynomial hierarchy at the second level (Karp-Lipton theorem), and $\mathbf{PSPACE} \subseteq \mathbf{NP/poly}$ would imply a collapse at the third level, since already $\mathbf{coNP} \subseteq \mathbf{NP/poly}$ would do so [21].

Proposition 30. *There is no polynomial-size translation from $\mathbf{NPDDL}$ into $\mathbf{NNFAT}$ unless $\mathbf{NP} \subseteq \mathbf{P/poly}$ holds.*

PROOF. We use the action description β_n that was introduced in Section 5; the size of β_n is clearly polynomial in n.
Assume that for every $\mathbf{NPDDL}$ action description α_n there is an equivalent $\mathbf{NNFAT}$ action description α'_n of size polynomial in that of α_n. In particular, there is an $\mathbf{NNFAT}$ action description β'_n of size polynomial in n which is equivalent to β_n. Then the following is a nonuniform polynomial-time algorithm for the 3-SAT problem; given a formula φ in 3-CNF over n variables:

1. encode φ into a state $s(\varphi)$ over the set of variables Q_n as in Notation 22;
2. decide whether $s(\varphi)$ is a β'_n-successor of $\emptyset$;
3. claim that φ is satisfiable if the answer is positive, otherwise claim that φ is unsatisfiable.

All steps are polynomial-time (Proposition 20), the algorithm is correct (Lemma 24), and the algorithm depends only on the number of variables in φ (which is polynomially related to the size of φ), hence this is indeed a nonuniform polynomial time algorithm for 3-SAT. Since 3-SAT is $\mathbf{NP}$-complete, we get $\mathbf{NP} \subseteq \mathbf{P/poly}$. □

We finally consider the relative succinctness of $\mathbf{DL\text{-}PPA}$ and $\mathbf{NPDDL}_{\mathrm{seq}}$. Since model checking in $\mathbf{DL\text{-}PPA}$ is $\mathbf{PSPACE}$-complete, there can be no polynomial time translation from $\mathbf{DL\text{-}PPA}$ to $\mathbf{NPDDL}_{\mathrm{seq}}$ unless $\mathbf{PSPACE} = \mathbf{NP}$. However, we will prove a stronger result.
For this, we use the problem of deciding whether a QBF formula is valid, for QBFs restricted to be of the form $\Phi = \forall x_1 \exists x_2 \ldots \forall x_{2n-1} \exists x_{2n} \varphi$, with φ a 3-CNF formula and $\mathrm{V}(\varphi) \subseteq X_{2n} = \{x_1, \ldots, x_{2n}\}$; clearly, deciding validity is as hard for such formulas (hereafter called "normalized QBFs") as for unrestricted QBFs, and hence it is $\mathbf{PSPACE}$-complete.
For all $n \in \mathbb{N}$, we define the $\mathbf{DL\text{-}PPA}$ action description $\langle \delta_{2n}, X_{2n} \cup Q_{2n} \cup \{r\} \rangle$, where Q_{2n} is as in Notation 22, r is a fresh variable (standing for "result"), and δ_{2n} is defined to be

$$\begin{aligned} r \quad \leftarrow \quad & \big([+x_1 \cup -x_1]\langle +x_2 \cup -x_2\rangle \\ & \ldots \\ & [+x_{2n-1} \cup -x_{2n-1}]\langle +x_{2n} \cup -x_{2n}\rangle \psi_{2n}\big) \end{aligned}$$

with $\psi_{2n} = \bigwedge_{q_i \in Q_{2n}} \big(q_i \rightarrow (\bigvee_{x \in \gamma_i} x \vee \bigvee_{\neg x \in \gamma_i} \neg x)\big)$ (rewritten without $\wedge$ nor $\rightarrow$ in polynomial time). Observe that the size of δ_{2n} is polynomial in n.

Lemma 31. *Let Φ be a normalized QBF over the set of variables $X_{2n} = \{x_1, \ldots, x_{2n}\}$. Then Φ is valid if and only if $s(\varphi) \cup \{r\}$ is a δ_{2n}-successor of $s(\varphi)$.*

PROOF. By the semantics of $\mathbf{DL\text{-}PPA}$, the modality $[+x_i \cup -x_i]$ mimicks exactly the quantification $\forall x_i$, and $\langle +x_i \cup -x_i \rangle$ mimicks exactly $\exists x_i$. On the other hand, it is easy to see that an assignment s_X to X_{2n} is a model of φ if and only if $s_X \cup s(\varphi)$ is a model of ψ_{2n}. It follows that Φ is valid if and only if $[+x_1 \cup -x_1]\langle +x_2 \cup -x_2\rangle \ldots [+x_{2n-1} \cup -x_{2n-1}]\langle +x_{2n} \cup -x_{2n}\rangle \psi_{2n}$ is true in $s(\varphi)$ and hence, that Φ is valid if and only if δ_{2n} assigns r to $\top$ when run in $s(\varphi)$, which finishes the proof. □

Using δ_{2n}, the proof of the following result is parallel to that of Proposition 30.

Proposition 32. *There is no polynomial-size translation from $\mathbf{DL\text{-}PPA}$ into $\mathbf{NPDDL}_{\mathrm{seq}}$ unless $\mathbf{PSPACE} \subseteq \mathbf{NP/poly}$ holds.*

PROOF. Assume that there is a polynomial-size translation from $\mathbf{DL\text{-}PPA}$ into $\mathbf{NPDDL}_{\mathrm{seq}}$, and for all n, let δ'_{2n} be an $\mathbf{NPDDL}_{\mathrm{seq}}$ description equivalent to δ_{2n} and of size polynomial in that of δ_{2n}, hence in n. Then the following is a nonuniform nondeterministic polynomial-time

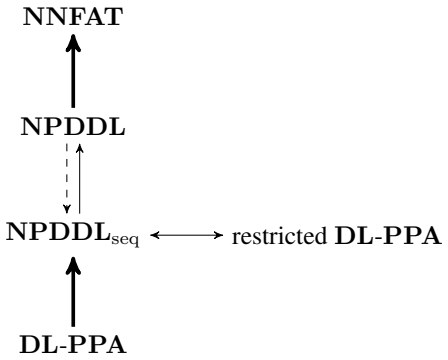


Figure 7.1: Succinctness relations between the languages. A thick arrow from L to L' means $L \prec L'$, a thin line means $L \preceq L'$, and a dashed arrow means that it is still unknown whether $L \preceq L'$.

algorithm for the problem of deciding the validity of a normalized QBF; given a normalized QBF over $2n$ variables, with matrix φ:

1. encode φ into $s(\varphi)$,
2. decide whether $s(\varphi) \cup \{r\}$ is a δ'_{2n}-successor of $s(\varphi)$,
3. claim that Φ is valid if the answer is positive, otherwise claim that Φ is not valid.

The steps are all feasible in deterministic or nondeterministic polynomial time (Proposition 21), the algorithm is correct by Lemma 31, and δ_{2n} depends only on the number of variables of Φ, hence this is indeed a nonuniform nondeterministic polynomial-time algorithm for deciding the validity of a normalized QBF. Since this is a **PSPACE**-complete problem, we conclude $\mathbf{PSPACE} \subseteq \mathbf{NP/poly}$. □

7 Conclusion

We have studied the complexity of deciding whether a state is a successor of another one through a given action, and the relative succinctness of three languages which are suitable for specifying planning tasks and actions. We have shown that deciding successorship is polynomial-time solvable for **NNFAT**, **NP**-complete for **NPDDL**, **NPDDL**$_{\text{seq}}$, and restricted **DL-PPA**, and **PSPACE**-complete for **DL-PPA**. The succinctness results agree with the intuition that the languages which are more succinct also have harder queries; the relationships which we have shown are represented on Figure 7.1.

An examination of the proof of Proposition 32 reveals that the reasons for **DL-PPA** being strictly more succinct than **NPDDL**$_{\text{seq}}$ are the modal operators. Our mid-term goal is to investigate complexity of queries and succinctness in a more systematic way, for languages constructed using combinations of features like the sequence operator, modalities, Kleene star, parallel execution, etc. For example, we want to try to find out whether **DL-PPA** without the Kleene star is strictly less succinct than **DL-PPA** (because until now the only elimination of $*$ that we know requires exponential space). Another interesting question is whether **NPDDL**$_{\text{seq}}$ is strictly more succinct than **NPDDL**, because IS-SUCC is **NP**-complete for both of them.

Acknowledgements

This work has been supported by the French National Research Agency (ANR) through project PING/ACK (ANR-18-CE40-0011).

References

[1] Alexandre Albore, Héctor Palacios, and Hector Geffner. Compiling uncertainty away in non-deterministic conformant planning. In *Proc. 19th European Conference on Artificial Intelligence (ECAI 2010)*, volume 215, pages 465–470, 2010.

[2] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[3] Christer Bäckström and Peter Jonsson. Algorithms and limits for compact plan representations. *Journal of Artificial Intelligence Research*, 44:141–177, 2012.

[4] Philippe Balbiani, Andreas Herzig, François Schwarzentruber, and Nicolas Troquard. DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE. *arXiv preprint arXiv:1411.7825*, 2014.

[5] Philippe Balbiani, Andreas Herzig, and Nicolas Troquard. Dynamic logic of propositional assignments: a well-behaved variant of PDL. In *Proc. 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LiCS 2013)*, pages 143–152, 2013.

[6] Piergiorgio Bertoli, Alessandro Cimatti, Ugo Dal Lago, and Marco Pistore. Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In *Proc. ICAPS 2003 Workshop on PDDL*, 2003.

[7] Daniel Bryce, Subbarao Kambhampati, and David E. Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[8] Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[9] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[10] Maria Fox and Derek Long. The third international planning competition: Temporal and metric planning. In *Proc. 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002)*, pages 333–335, 2002.

[11] 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.

[12] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[13] Tomas Geffner and Hector Geffner. Compact policies for fully observable non-deterministic planning as SAT. In *Proc. 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pages 88–96, 2018.

[14] Andreas Herzig, Frédéric Maris, and Julien Vianey. Dynamic logic of parallel propositional assignments and its applications to planning. In *Proc. 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 5576–5582, 2019.

[15] Drew McDermott. PDDL–the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. Available at: `www.cs.yale.edu/homes/dvm` (consulted on 2020/03/16).

[16] Christian J. Muise, Sheila A. McIlraith, and Vaishak Belle. Non-deterministic planning with conditional effects. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pages 370—374, 2014.

[17] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

[18] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, pages 359–380. Academic Press Professional, 1991.

[19] Jussi Rintanen. Complexity of planning with partial observability. In *Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 345–354, 2004.

[20] Son Thanh To, Tran Cao Son, and Enrico Pontelli. A generic approach to planning in the presence of incomplete information: Theory and implementation. *Artificial Intelligence*, 227:1–51, 2015.

[21] Chee K Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical computer science*, 26(3):287–300, 1983.

Sur le principe d'optimalité de Bellman pour les zs-POSG

Olivier Buffet[1] Jilles Dibangoye[2] Aurélien Delage[1,2] Abdallah Saffidine[3] Vincent Thomas[1]

[1] Université de Lorraine, CNRS, INRIA, LORIA, F-54000 Nancy
[2] Université de Lyon, INSA Lyon and Inria, CITI, F-69000 Lyon
[3] University of New South Wales, Sydney, Australie

(prenom.nom@inria.fr|abdallahs[@]cse.unsw.edu.au)

Résumé

*De nombreux problèmes de prise de décision séquentielle sont résolus efficacement en exploitant le principe d'optimalité de Bellman, c'est-à-dire l'imbrication récursive de sous-problèmes dans le problème original. Nous montrons ici qu'il peut être appliqué aux jeux stochastiques partiellement observables à 2 joueurs et somme nulle (zs-POSG) en (i) prenant le point de vue d'un planificateur central, qui ne peut raisonner que sur une statistique suffisante appelée état d'occupation, et (ii) transformant de tels problèmes en des jeux de Markov dans l'espace des «états d'occupation» à somme nulle (zs-OMG). Ensuite, en exploitant des propriétés de Lipschitz-continuité de la fonction de valeur optimale, on peut dériver une version de l'algorithme HSVI (*Heuristic Search Value Iteration*) qui trouve un ϵ-équilibre de Nash en temps fini.*

Mots Clef

POSG ; jeux stochastiques partiellement observables ; principe d'optimalité de Bellman ; Heuristic Search Value Iteration.

Abstract

Many non-trivial sequential decision-making problems are efficiently solved by exploiting Bellman's optimality principle, i.e.*, the recursive nesting of sub-problems within the original problem. Here we show how it can apply to (infinite horizon) 2-player zero-sum partially observable stochastic games (zs-POSGs) by (i) taking a central planner's viewpoint, which can only reason on a sufficient statistic called occupancy state, and (ii) turning such problems into zero-sum occupancy Markov games (zs-OMGs). Then, exploiting Lipschitz-continuity properties of the optimal value function, one can derive a version of the HSVI algorithm (Heuristic Search Value Iteration) that provably finds an ϵ-Nash equilibrium in finite time.*

Keywords

POSG ; partially observable stochastic games ; Bellman's optimality principle ; Heuristic Search Value Iteration.
[Note : Cet article résume un document à paraître [6].]

1 Introduction

Le principe d'optimalité de Bellman [3] a conduit à des solveurs de référence pour de nombreux problèmes de prise de décision séquentielle non-triviaux, supposant une observabilité partielle [19], des critères multi-objectifs [22, 16], des agents collaborant, par exemple modélisés par des processus de décision markoviens partiellement observables décentralisés (Dec-POMDP) [12, 23, 10], ou certains jeux non-collaboratifs (à commencer par le travail précurseur de Shapley [20], voir aussi [7]). Dans chacun de ces cadres ce principe exploite l'imbrication récursive de sous-problèmes dans le problème original. La question reste ouverte de savoir si — et comment — ce principe pourrait être appliqué aux jeux à information imparfaite, lesquels sont rencontrés dans des applications diverses telles que le Poker [15] ou les jeux de sécurité [1]. Ce travaille répond à cette question dans le cadre des jeux stochastiques partiellement observables à 2-joueurs et somme nulle (zs-POSG), *c.-à-d.* des jeux à information imparfaite, actions simultanées, mémoire parfaite (*perfect recall*), récompenses atténuées et horizon temporel potentiellement infini.

2 État de l'art

Comme les POSG et Dec-POMDP généraux, les zs-POSG à horizon infini sont indécidables, et leurs approximations à horizon fini sont dans NEXP [17, 4]. Les techniques de résolution pour POSG à horizon fini, ou autres jeux à information imparfaite qui peuvent être formulés comme des jeux en forme extensive (EFG), résolvent typiquement un programme linéaire [21], par exemple dérivé du jeu en forme normale équivalent, ou emploient un mécanisme de minimisation du regret dédié [25, 5]. Ils ne reposent donc pas sur le principe d'optimalité de Bellman, sauf (i) une approche par programmation dynamique qui ne fait que construire des ensembles de solutions non dominées [12], (ii) dans les problèmes collaboratifs (POMDP décentralisés), en adoptant le point de vue d'un planificateur central [23, 10], et (iii) pour des cadres (majoritairement à 2 joueurs et somme nulle) sous des hypothèses d'observabilité telles que l'on peut raisonner sur les croyances des joueurs [11, 8, 2, 14, 9, 13]. Ici, nous ne faisons pas d'hypothèses autres que le jeu étant

à 2 joueurs et somme nulle, en particulier concernant l'observabilité de l'état et des actions.

3 Des zs-POSG aux zs-OMG

Comme dans nombre de solveurs de Dec-POMDP, notre approche adopte le point de vue non d'un joueur, mais d'un planificateur central (hors-ligne) qui prescrit une stratégie individuelle à chaque joueur [23] en raisonnant sur les profils de stratégies partielles exécutés de $t = 0$ à τ. Le planificateur fait ainsi face à un jeu dont la dynamique est déterministe, puisqu'une stratégie partielle évolue de manière déterministe quand on la complète d'une paire de règles de décision sur un pas de temps, et dans lequel les actions des joueurs (prises non pas dans les ensembles d'actions de départ, mais parmi les règles de décision choisies par l'un et l'autre joueur) sont publiques. Ainsi, le profil de stratégies partielles courant est toujours connu des deux joueurs.

Validité du principe d'optimalité de Bellman Une statistique introduite initialement pour résoudre les Dec-POMDP, l'*état d'occupation* o_τ [10] — *c.-à-d.* la distribution de probabilité sur (i) l'état du système et (ii) l'historique d'action-observation jointe des deux joueurs étant donnée un profil de stratégies partielles — est alors employée pour démontrer que

1. étant donné un profil de stratégies partielles, on peut trouver, à l'aide de l'état d'occupation correspondant, un profil de stratégies optimal des pas de temps τ à H (l'horizon temporel fini) ;
2. toute solution optimale de ce jeu est construite récursivement sur des solutions optimales de sous-problèmes imbriqués, ce qui correspond au principe d'optimalité de Bellman ; et
3. l'état d'occupation est une statistique suffisante (en remplacement du profil de stratégies partielles) pour résoudre de manière optimale le jeu auquel est confronté le planificateur central.

Dans la suite, nous dénotons $V^*_\tau(o_\tau)$ la valeur optimale (à l'équilibre de Nash) pour le sous-problème rencontré dans l'état d'occupation o_τ, et appelons $V^*_\tau(\cdot)$ la *fonction de valeur optimale*.

Des jeux en forme anormale Le jeu en question est appelé un *occupancy Markov Game* (OMG). [1] Un OMG n'est toutefois pas un jeu de Markov standard, non seulement parce que l'espace d'états est continu et la dynamique déterministe, mais aussi parce que les *jeux locaux* dans les états d'occupation rencontrés, où l'on optimise le profil de règles de décision immédiates $\boldsymbol{\beta}_\tau$, ne sont pas des jeux en forme normale, de sorte qu'ils ne peuvent être résolus via un LP comme pour les jeux en forme normale à 2 joueurs et somme nulle. On peut toutefois raisonner sur des *sous-jeux*, dans lesquels on optimise non plus les seules règles de décision immédiate, mais toutes jusqu'à $H-1$, sous-jeux qui sont, eux, équivalents à des jeux en forme normale. Cela permet de prouver que les valeurs maximin et minimax des jeux locaux en forme **a**normale sont égales, correspondant donc à la valeur unique des équilibres de Nash, et induisent un profil de stratégies en équilibre de Nash.

La section suivante introduit des ingrédients clefs qui rendent possible la résolution de zs-OMG en exploitant le principe d'optimalité de Bellman.

4 Exploiter la structure de V^* et Q^*

Deux difficultés qui nous empêchent pour l'instant de résoudre des zs-OMG sont que :

1. à cause des espaces continus d'états et d'actions, on ne peut résoudre le problème en explorant l'infiniment-ramifié (*infinitely-branching*) arbre des futurs possibles sans capacités de généralisation ; et
2. il nous manque une solution pour résoudre un jeu en forme anormale donné.

Dans la suite, nous exploitons les propriétés des fonctions de valeur optimales V^* et Q^* pour traiter ces deux difficultés.

Lipschitz-continuité de V^* et Q^* Les propriétés de linéarité et de Lipschitz-continuité (LC) des fonctions de transition et de récompense du zs-OMG permettent de démontrer que, dans le cas d'un horizon temporel fini,

— la fonction de valeur optimale $V^*_\tau(o_\tau)$ est LC dans l'espace des états d'occupation ; et

— la fonction d'action-valeur optimale $Q^*_\tau(o_\tau, \beta^1_\tau, \beta^2_\tau)$ est LC dans les espaces des états d'occupation o_τ comme des règles de décision individuelles β^1_τ et β^2_τ.

Résolution de jeux en forme *a*normale $Q^*_\tau(o_\tau, \cdot, \cdot)$ définissant un jeu en forme anormale à résoudre en o_τ, ses propriétés de Lipschitz-continuité permettent d'envisager une procédure d'optimisation bi-niveau reposant sur un algorithme tel que DOO (*Deterministic Optimistic Optimization*) de Munos [18] pour résoudre de manière ϵ-optimale les problèmes maximin et minimax.

Approximations de V^* La Lipschitz-continuité de V^* permet de dériver des approximateurs à base de points majorant et minorant, à l'aide de cônes pointant respectivement vers le bas et vers le haut. (Imaginer une mâchoire de dents pointues.) Diverses relaxations du zs-POSG à résoudre peuvent alors être envisagées pour dériver les initialisations de ces approximateurs majorant et minorant.

5 HSVI pour zs-POSG

Enfin, la capacité de maintenir de tels approximateurs encadrant (en résolvant les jeux locaux à l'aide d'une optimisation bi-niveau globale) permet de décrire une variante de HSVI pour zs-OMG, donc zs-POSG. Rappelons que HSVI repose sur (i) la génération de trajectoires au cours

1. Nous préférons «jeu de Markov» à «jeu stochastique» à cause du déterminisme de la dynamique.

desquelles chaque joueur agit au mieux d'après l'approximateur qui est optimiste pour lui, et (ii) la mise-à-jour au fur et à mesure les approximateurs jusqu'à atteindre une précision suffisante.
Un critère d'arrêt inspiré par [13] permet de compenser le facteur de branchement infini, et ainsi de démontrer que l'algorithme converge en temps fini vers une solution ϵ-optimale malgré les espaces d'état (d'occupation) et d'action continus.

6 Discussion

Le présent travail démontre théoriquement que l'on peut résoudre des zs-POSG en exploitant la propriété d'optimalité de Bellman. Certains détails d'implémentation comme l'optimisation bi-niveau, l'élagage de cones, ou la possibilité d'utiliser des techniques de compression de l'état d'occupation doivent toutefois être discutés plus avant afin de pouvoir envisager une étude expérimentale de l'approche. En outre, une piste d'amélioration en cours d'étude est la possibilité d'exploiter les propriétés de concavité-convexité de la fonction de valeur optimale démontrées par Wiggers et al. [24] pour obtenir à la fois des approximateurs plus fins et des optimisations bi-niveaux plus efficaces.

Références

[1] N. Basilico, G. De Nittis et N. Gatti : A security game combining patrolling and alarm–triggered responses under spatial and detection uncertainties. Dans *AAAI-16*, 2016.

[2] A. Basu et L. Stettner : Finite- and infinite-horizon Shapley games with nonsymmetric partial observation. *SIAM Journal on Control and Optimization*, 53(6): 3584–3619, 2015.

[3] R. Bellman : On the theory of dynamic programming. *PNAS*, 38:716–719, 1952.

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

[5] N. Brown et T. Sandholm : Superhuman AI for heads-up no-limit poker : Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[6] O. Buffet, J. Dibangoye, A. Delage, A. Saffidine et V. Thomas : On Bellman's optimality principle for zs-POSGs, 2020. (à paraître).

[7] O. Buffet, J. Dibangoye, A. Saffidine et V. Thomas : Heuristic search value iteration for zero-sum stochastic games. *IEEE Transactions on Games*, 2020. (à paraître).

[8] K. Chatterjee et L. Doyen : Partial-observation stochastic games : How to win when belief fails. vol. 15, p. 16, 2014.

[9] H. L. Cole et N. Kocherlakota : Dynamic games with hidden actions and hidden states. *Journal of Economic Theory*, 98(1):114–126, 2001.

[10] J. Dibangoye, C. Amato, O. Buffet et F. Charpillet : Optimally solving Dec-POMDPs as continuous-state MDPs. *JAIR*, 55:443–497, 2016.

[11] M. K. Ghosh, D. McDonald et S. Sinha : Zero-sum stochastic games with partial information. *Journal of Optimization Theory and Applications*, 121(1):99–118, avr. 2004.

[12] E. A. Hansen, D. Bernstein et S. Zilberstein : Dynamic programming for partially observable stochastic games. Dans *AAAI-04*, 2004.

[13] K. Horák et B. Bošanský : Solving partially observable stochastic games with public observations. Dans *AAAI-19*, p. 2029–2036, 2019.

[14] K. Horák, B. Bošanský et M. Pěchouček : Heuristic search value iteration for one-sided partially observable stochastic games. Dans *AAAI-17*, p. 558–564, 2017.

[15] H. W. Kuhn : Simplified two-person Poker. Dans H. W. Kuhn et A. W. Tucker, éds : *Contributions to the Theory of Games*, vol. 1. Princeton University Press, 1950.

[16] E. Machuca : An analysis of multiobjective search algorithms and heuristics. Dans *IJCAI-11*, 2011.

[17] O. Madani, S. Hanks et A. Condon : On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. Dans *AAAI-99*, 1999.

[18] R. Munos : From bandits to Monte-Carlo Tree Search : The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, 7(1):1–130, 2014.

[19] K. Åström : Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174 – 205, 1965.

[20] L. S. Shapley : Stochastic games. *PNAS*, 39(10):1095–1100, 1953.

[21] Y. Shoham et K. Leyton-Brown : *Multiagent Systems : Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[22] B. S. Stewart et C. C. White, III : Multiobjective A*. *Journal of the ACM*, 38(4):775–814, oct. 1991.

[23] D. Szer, F. Charpillet et S. Zilberstein : MAA* : A heuristic search algorithm for solving decentralized POMDPs. Dans *UAI-05*, p. 576–583, 2005.

[24] A. Wiggers, F. Oliehoek et D. Roijers : Structure in the value function of two-player zero-sum games of incomplete information. Dans *ECAI-16*, p. 1628–1629, 2016.

[25] M. Zinkevich, M. Johanson, M. Bowling et C. Piccione : Regret minimization in games with incomplete information. Dans *NIPS-07*, 2007.

End-to-end learning of reusable skills through intrinsic motivation

A. Aubret[1] L. Matignon[2] S. Hassas[1]

[1] Univ Lyon, Université Lyon 1, CNRS, LIRIS F-69622, Villeurbanne, France

arthur.aubret@univ-lyon1.fr

June 23, 2020

Abstract

Humans beings do not think about the muscles they contract but rather take high level decisions such as walking or running. Taking inspiration from developmental learning, we present a novel reinforcement learning architecture which hierarchically learns and represents self-generated skills in an end-to-end way. With this architecture, an agent focuses only on task-rewarded skills while keeping the learning process of skills bottom-up. This bottom-up approach allows to learn skills independently from extrinsic reward and transferable across tasks. To do that, we combine a previously defined mutual information objective with a novel curriculum learning algorithm, creating an unlimited and explorable tree of skills. We test our agent on a simple gridword environment to understand and visualize how the agent distinguishes between its skills. Then we show that our approach can scale on more difficult MuJoCo environments in which our agent is able to build a representation of skills which facilitates transfer learning.

Keywords

Intrinsic motivation, curriculum learning, developmental learning, reinforcement learning.

1 Introduction

In reinforcement learning (RL), an agent learns by trial-and-error to maximize the expected rewards obtained from actions performed in its environment [41]. However, many RL agents usually strive to achieve one goal using only low-level actions. In contrast, as humans being, when we want to go to work, we do not think about every muscle we contract in order to move; we just take abstract decisions such as *Go to work*. Low-level behaviors such as how to walk are already learned and we do not need to think about them. Learning to walk is a classical example of babies developmental learning, which refers to the ability of an agent to spontaneously explore its environment and acquire new skills [9]. Babies do not try to get walking behaviors all at once, but rather first learn to move their legs, to crawl, to stand up, and then, eventually, to walk. They are **intrinsically motivated** since they act for the inherent satisfaction of learning new skills [37] rather than for an **extrinsic reward** assigned by the environment.

Several works are interested in learning abstract actions, also named **skills** or **options** [42], in the framework of deep reinforcement learning (DRL) [4]. Skills can be learned with extrinsic rewards [5], which facilitates the credit assignment [42]. In contrast, if one learns skills with intrinsic motivation, the learning process becomes bottom-up [26], i.e. the agent learns skills before getting extrinsic rewards. When learning is bottom-up, the agent commits to a time-extended skill and avoids the usual wanderlust due to the lack, or the sparsity, of extrinsic rewards. Therefore, it can significantly improve exploration [27, 34]. In addition, these skills can be used for different tasks, emphasizing their potential for transfer learning [43]. These properties make intrinsic motivation attractive in a *continual learning* framework, which is the ability of the agent to acquire, retain and reuse its knowledge over a lifetime [45].

Several works recently proposed to intrinsically learn such skills using a diversity heuristic [14, 2], such that different states are covered by the learned skills. Yet several issues remain: 1- the agent is often limited in the number of learned skills or requires *curriculum learning* [2]; 2- most skills target uninteresting parts of the environment relatively to some tasks; thereby it requires prior knowledge about which features to diversify [14]; 3- the agent suffers from catastrophic forgetting when it tries to learn a task while learning skills [14]; 4- discrete time-extended skills used in a hierarchical setting are often sub-optimal for a task. With diversity heuristic, skills are indeed not expressive enough to efficiently target a goal [14, 2].

In this paper, we propose to address these four issues so as to **improve the approaches for continually learning increasingly difficult skills with diversity heuristics**. We introduce ELSIM (End-to-ended Learning of reusable Skills through Intrinsic Motivation), a method for learning representations of skills in a bottom-up way. The agent autonomously builds a tree of abstract skills where each skill is a refinement of its parent. First of all, skills are learned independently from the tasks but along with tasks; it guarantees they can be easily transferred to other tasks and may help the agent to explore its environment. We use the op-

timization function defined in [14] which guarantees that states targeted by a skill are close to each other. Secondly, the agent selects a skill to refine with extrinsic or intrinsic rewards, and learns new sub-skills; it ensures that the agent learns specific skills useful for tasks through an intelligent *curriculum*, among millions of possible skills.
Our approach contrasts with existing approaches which either bias skills towards a task [5], reducing the possibilities for transfer learning, or learn skills during pretraining [14]. We believe our paradigm, by removing the requirement of a *developmental period* [29] (which is just an unsupervised pretraining), makes naturally compatible developmental learning and *lifelong learning*. Therefore, we emphasize three properties of our ELSIM method. 1- **Learning is bottom-up**: the agent does not require an expert supervision to expand the set of skills. It can use its skills to solve different sequentially presented tasks or to explore its environment. 2- **Learning is end-to-end**: the agent never stops training and keeps expanding its tree of skills. It gradually self-improves and avoids catastrophic forgetting. 3- **Learning is focused**: the agent only learns skills useful for its high-level extrinsic/intrinsic objectives when provided.
Our contributions are the following: we introduce a new curriculum algorithm based on an adaptation of diversity-based skill learning methods. Our objective is not to be competitive when the agent learns one specific goal, but **to learn useful and reusable skills along with sequentially presented goals in an end-to-end fashion**. We show experimentally that ELSIM achieves **good asymptotic performance** on several single-task benchmarks, **improves exploration** over standard DRL algorithms and manages to easily **reuse its skills**. Thus, this is a step towards *lifelong learning* agents.
This paper is organized as follows. First, we introduce the concepts used in ELSIM, especially diversity-based intrinsic motivation (Section 2). In Section 3, the core of our method is presented. Then, we explain and visualize how ELSIM works on simple gridworlds and compare its performances with state-of-the-art DRL algorithms on single and sequentially presented tasks learning (Section 4). In Section 5, we detail how ELSIM relates to existing works. Finally, in Section 6, we take a step back and discuss ELSIM.

2 Background

2.1 Reinforcement learning

A Markov decision process (MDP) [36] is defined by a set of possible states S; a set of possible actions A; a transition function $P : S \times A \times S \to \mathbb{P}(s'|s,a)$ with $a \in A$ and $s, s' \in S$; a reward function $R : S \times A \times S \to \mathbb{R}$; the initial distribution of states $\rho_0 : S \to [0;1]$. A stochastic policy π maps states to probabilities over actions in order to maximize the discounted cumulative reward defined by $\varsigma_t = [\sum_{t=0}^{\infty} \gamma^t r_t]$ where $\gamma \in [0,1]$ is the discount factor. In order to find the action maximizing ς in a state s, it is common to maximize the expected discounted gain following a policy π from a state-action tuple defined by:

$$Q_\pi(s,a) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(s_t) \\ s_{t+1} \sim P(s_{t+1}|s_t,a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \right] \tag{1}$$

where $s_0 = s$, $a_0 = a$. To compute this value, it is possible to use the Bellman Equation [41].

2.2 Obtaining diverse skills through mutual information objective

We characterize a **skill** by its intra-skill policy; thereby a skill is a mapping of states to probabilities over actions.
One way to learn, without extrinsic rewards, a set of different skills along with their intra-skill policies is to use an objective based on mutual information (MI).
In [14], learned skills should be as **diverse** as possible (different skills should visit different states) and **distinguishable** (it should be possible to infer the skill from the states visited by the intra-skill policy). It follows that the learning process is 4-step with two learning parts [14]: 1- the agent samples one skill from an uniform distribution; 2- the agent executes the skill by following the corresponding intra-skill policy (randomly initialized at the beginning); 3- a discriminator learns to categorize the resulting states to the assigned skill; 4- at the same time, these approximations reward intra-skill policies (cf. Equation (5)).
The global objective can be formalized as maximizing the MI between the set of skills G and states S' visited by intra-skill policies, defined by [18]:

$$I(G;S') = \mathbb{H}(G) - \mathbb{H}(G|S') \tag{2}$$

$$= \mathop{\mathbb{E}}_{\substack{g \sim p(g) \\ s' \sim p(s'|\pi_\theta^g, s)}} [\log p(g|s') - \log p(g)] \tag{3}$$

where π_θ^g is the intra-skill policy of $g \in G$ and is parameterized by θ; $p(g)$ is the distribution of skills the agent samples on; and $p(g|s')$ is the probability to infer g knowing the next state s' and intra-skill policies. This MI quantifies the reduction in the uncertainty of G due to the knowledge of S'. By maximizing it, states visited by an intra-skill policy have to be informative of the given skill.
A bound on the MI can be used as an approximation to avoid the difficulty to compute $p(g|s')$[8, 18] :

$$I(G,S') \geq \mathop{\mathbb{E}}_{\substack{g \sim p(g) \\ s' \sim p(s'|\pi_\theta^g, s)}} [\log q_\omega(g|s') - \log p(g)] \tag{4}$$

where $q_\omega(g|s')$ is the **discriminator** approximating $p(g|s')$. In our case, the discriminator is a neural network parameterized by ω. q_ω minimizes the standard cross-entropy $-\mathbb{E}_{g \sim p(g|s')} \log q_\omega(g|s')$ where $s' \sim \pi_\theta^g$.
To discover skills, it is more efficient to set $p(g)$ to be uniform as it maximizes the entropy of G [14]. Using the uniform distribution, $\log p(g)$ is constant and can be removed

from Equation (4). It follows that one can maximize Equation (4) using an intrinsic reward to learn the intra-skill policy of a skill $g \in G$ [14]:

$$r^g(s') = \log q_\omega(g|s'). \quad (5)$$

Similarly to [14], we use an additional entropy term to encourage the diversity of covered states. In practice, this bonus is maximized through the use of DRL algorithms: Soft Actor Critic (SAC) [20] for continuous action space and Deep Q network (DQN) with Boltzmann exploration [30] for discrete one.

3 Method

In this section, we first give an overview of our method and then detail the building of the tree of skills, the learning of the skill policy, the selection of the skill to refine and how ELSIM integrates this in an end-to-end framework.

3.1 Overview: building a tree of skills

To get both bottom-up skills and interesting skills relatively to some tasks, our agent has to choose the skills to improve thanks to the extrinsic rewards, but we want that our agent improves its skills without extrinsic rewards. The agent starts by learning a discrete set of diverse and distinguishable skills using the method presented in Section 2.2. Once the agent clearly distinguishes these skills using the covered skill-conditioned states with its discriminator, it splits them into new sub-skills. For instance, for a creature provided with proprioceptive data, a *moving forward* skill could be separated into *running* and *walking*. The agent only trains on sub-skills for which the parent skill is useful for the global task. Thus it incrementally refines the skills it needs to accomplish its current task. If the agent strives to sprint, it will select the skill that provides the greater speed. The agent repeats the splitting procedure until its intra-skill policy either reach the maximum number of splits or become too deterministic to be refined. The **hierarchy of skills** is maintained using a tree where each node refers to an abstract skill that has been split and each leaf is a skill being learned. We formalize the hierarchy using sequence of letters where a letter's value is assigned to each node:

- The set of skills G is the set of leaf nodes. A skill $g \in G$ is represented by a sequence of $k+1$ letters : $g = (l^0, l^1, ..., l^k)$. When g is split, a letter is added to the sequence of its new sub-skills. For instance, the skill $g = (l^0 = 0, l^1 = 1)$ can be split into two sub-skills $(l^0 = 0, l^1 = 1, l^2 = 0)$ and $(l^0 = 0, l^1 = 1, l^2 = 1)$.
- The vocabulary V refers to the values which can be assigned to a letter. For example, to refine a skill into 4 sub-skills, we should define $V = \{0, 1, 2, 3\}$.
- The length $L(g)$ of a skill is the number of letters it contains. Note that the length of a skill is always larger than its parent's.
- $l^{:k}$ is the sequence of letters preceding l^k (excluded).

We use two kind of policies: the first are the **intra-skill policies**. The learning of these intra-skill policies is described in Section 3.2. The second type of policy is task-dependent and responsible to choose which skill to execute; we call it the **tree-policy** (see Section 3.3).

3.2 Learning intra-skill policies

In this section, we detail how intra-skill policies are learned. We adapt the method presented in Section 2.2 to our hierarchical skills context. Two processes are simultaneously trained to obtain diverse skills: the intra-skill policies learn to maximize the intrinsic reward (cf. Equation (5)), which requires to learn a discriminator $q_\omega(g|s')$. Given our hierarchic skills, we can formulate the probability inferred by the discriminator as a product of the probabilities of achieving each letter of g knowing the sequence of preceding letters, by applying the chain rule:

$$\begin{aligned} r^g(s') &= \log q_\omega(g|s') = \log q_\omega(l^0, l^1, \dots, l^k|s') \\ &= \sum_{i=0}^{k} \log q_\omega(l^i|s', l^{:i}). \end{aligned} \quad (6)$$

Gathering this value is difficult and requires an efficient discriminator q_ω. As it will be explained in Section 3.4, in practice, we use **one different discriminator for each node** of our tree: $\forall i,\ q_\omega(l^i|s', l^{:i}) \equiv q_\omega^{:i}(l^i|s')$.
For instance, if $|V| = 2$, one discriminator $q_\omega^{\emptyset}$ will be used to discriminate $(l^0 = 0)$ and $(l^0 = 1)$ but an other one, $q_\omega^{l^0=0}$ will discriminate $(l^0 = 0, l^1 = 0)$ and $(l^0 = 0, l^1 = 1)$.
It would be difficult for the discriminators to learn over all letters at once; the agent would gather states for several inter-level discriminators at the same time and a discriminator would not know which part of the gathered states it should focus on. This is due to the fact that discriminators and intra-skill policies simultaneously train. Furthermore, there are millions of possible combinations of letters when the maximum size of sequence is large. We do not want to learn them all. To address these issues, we introduce **a new curriculum learning algorithm that refines a skill only when it is distinguishable**. When discriminators successfully learn, they progressively extends the sequence of letters; in fact, we split a skill (add a letter) only when its discriminator has managed to discriminate the values of its letter. Let's define the following probability:

$$p_{finish}^{:k}(l^k) = \mathbb{E}_{s_{final} \sim \pi^{:k+1}} \left[q_\omega^{:k}(l^k|s_{final}) \right]. \quad (7)$$

where s_{final} is the state reached by the intra-skill policy at the last timestep. We assume the discriminator $q_\omega^{:k}$ has finished to learn when: $\forall v \in V,\ p_{finish}^{:k}(l^k = v) \geq \delta$ where $\delta \in [0, 1]$ is an hyperparameter. Choosing a δ close to 1 ensures that the skill is learned, but an intra-skill policy always explores, thereby it may never reach an aver-

age probability of exactly 1; we found empirically that 0.9 works well.

To approximate Equation (7) for each letters' value v, we use an exponential moving average $p^{:k}_{finish}(l^k = v) = (1-\beta)p^{:k}_{finish}(l^k = v) + \beta q^{:k}_{\omega}(l^k = v|s_{final})$ where $s_{final} \sim \pi^{:k+1}$ and $\beta \in [0; 1]$. Since we use buffers of interactions (see Section 3.4), we entirely refill the buffer before the split.

Let us reconsider Equation (6). $\sum_{i=0}^{k-1} \log q_\omega(l^i|s', l^{:i})$ is the part of the reward assigned by the previously learned discriminators. It forces the skill to stay close to the states of its parent skills since this part of the reward is common to all the rewards of its parent skills. In contrast, $\log q_\omega(l^k|s', l^{:k})$ is the reward assigned by the discriminator that actively learns a new discrimination of the state space. Since the agent is constrained to stay inside the area of previous discriminators, the new discrimination is **uncorrelated** from previous parent discriminations. In practice, we increase the importance of previous discriminations with a hyper-parameter $\alpha \in \mathbb{R}$:

$$r^g(s') = \log q_\omega(l^k|s', l^{:k}) + \alpha \sum_{i=0}^{k-1} \log q_\omega(l^i|s', l^{:i}). \quad (8)$$

This hyper-parameter is important to prevent the agent to deviate from previously discriminated areas to learn more easily the new discrimination.

3.3 Learning which skill to execute and train

For each global objective, a stochastic policy, called *tree-policy* and noted π_T (with T the tree of skills), is responsible to choose the skill to train by navigating inside the tree at the beginning of a task-episode. This choice is critical in our setting: while expanding its tree of skills, the agent cannot learn to discriminate every leaf skill at the same time since discriminators need states resulting from the intra-skill policies. We propose to **choose the skill to refine according to its benefit in getting an other reward** (extrinsic or intrinsic), thereby ELSIM executes and learns only interesting skills (relatively to an additional reward).

To learn the *tree-policy*, we propose to model the tree of skills as an MDP solved with a Q-learning and Boltzmann exploration. The action space is the vocabulary V; the state space is the set of nodes, which include abstract and actual skills; the deterministic transition function is the next node selection; if the node is not a leaf, the reward function R_T is 0, else this is the discounted reward of the intra-skill policy executed in the environment divided by the maximal episode length. Each episode starts with the initial state as the root of the tree, the *tree-policy* selects the next nodes using Q-values. Each episode ends when a leaf node has been chosen, i.e. a skill for which all its letters has been selected; the last node is always chosen uniformly (see Section 3.4).

Let us roll out an example using the *tree-policy* displayed in Figure 1. The episode starts at the root of the tree; the *tree-policy* samples the first letter, for example it selects $l^0 = 0$. Until it reaches a leaf-node, it samples new letters, e.g. $l^1 = 1$ and $l^2 = 0$. The *tree-policy* has reached a leaf, thereby it will execute and learn the skill $(0, 1, 0)$. Then, the state-action tuple $((0, 1), (0))$ is rewarded with the scaled discounted reward of the task. This reward is propagated via the Q-learning update to previous state-action tuples $((\emptyset), (0))$ and $((0), (1))$ to orientate the *tree-policy* to $(0, 1, 0)$.

The MDP evolves during the learning process since new letters are progressively added. The Q-values of new skills are initialized with their parent Q-values. However, Equation (6) ensures that adding letters at the leaf of the tree monotonically increases Q-values of their parent nodes. The intuition is that, when splitting a skill, at least one of the child is equal or better than the skill of its parent relatively to the task. We experimentally show this in Section 4.2. The resulting curriculum can be summarized as follows: the tree will be small at the beginning, and will grow larger in the direction of feedbacks of the environment.

We now sum up the process of the *tree-policy*: 1-an agent runs an episode inside the MDP of skills; the sequence of actions represents a skill; 2- the agent executes the intra-skill policy of the skill; 3- the *tree-policy* is rewarded according to how well the intra-skill policy fits the task and the Q-learning applies. The full algorithm of the *tree-policy* is given in Appendix A.

3.4 Simultaneous training of the *tree-policy* and intra-skill policies

The MI objective requires the skill distribution to remain uniform (cf. Equation (5)), however that is not our case: the agent strives to avoid some useless skills while focusing on others. In our preliminary experiments, ignoring this leads us to catastrophic forgetting of the learned skills since discriminators forget how to categorize states of the skills they never learn on. To bypass this issue and sample uniformly, we assign to each node i of our tree a replay buffer containing interactions of the intra-skill policy with the environment, a RL algorithm and a discriminator (q^i_ω). At each split, intra-skill policies and buffers of a node are copied to its children; for the first node, its intra-skill policies are randomly initialized and its buffer is empty.

This way, the entire training is off-policy: the intra-skill policy fills the replay buffer while the discriminator and intra-skill policies learn from the interactions that are uniformly extracted from their buffers. We split the lifetime of a node into two phases: 1-the **learning phase** during which next letter's values are sampled uniformly; the *tree-policy* is uniform at this node; 2-the **exploitation phase** during which the *tree-policy* chooses letters with its Boltzmann policy (Section 3.3).

Then, at each step, the agent runs the *tree-policy* to select the discriminator in the learning phase that will learn. The discriminator samples a mini-batch of data from its children's (all leaves) buffers and learns on it. Then, all children intra-skill policies learn from the intrinsic feedback of

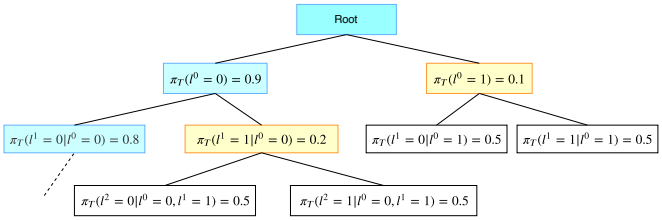


Figure 1: Representation of a part of the tree of skills with $|V| = 2$ and the value of *tree-policy* in each node. White nodes are actual leaves of the tree; the discriminator is inactive. Yellow nodes represent nodes for which the discriminator can not differentiate its sub-skills; the *tree-policy* samples uniformly. Nodes are blue when the discriminator can distinguish its sub-skills; the *tree-policy* samples using Q-values.

the same interactions, output by the selected discriminator and all its parents according to Equation (6).
Once a node enters the exploitation phase, an hyper-parameter η regulates the probability that each parent's discriminator learn on its children data. Their learning interactions are recursively sampled uniformly on their children. This post-exploration learning allows a node to expand its high-reward area. Without this mechanism, different uncovered states of the desired behaviour may be definitively attributed to different fuzzy skills, as shown in Section 4.1. The full learning algorithm is given in Appendix B. Figure 1 gives an example of a potential tree and how different phases coexist; the skills starting by $(0, 1)$ seem to be the most interesting for the task since each letter sampling probability is high. Skills $(0, 1, 0)$, $(0, 1, 1)$, $(1, 0)$ and $(1, 1)$ are being learned, therefore the sampling probability of their last values is uniform.

4 Experiments

The first objective of this section is to study the behavior of our ELSIM algorithm on basic gridworlds to make the visualization easier. The second purpose is to show that ELSIM can scale with high-dimensional environments. We also compare its performance with a non-hierarchical algorithm SAC [20] in a single task setting. Finally we show the potential of ELSIM for transfer learning.

4.1 Study of ELSIM in gridworlds

In this section, we analyze how skills are refined on simple gridworlds adapted from gym-minigrid [11]. Unless otherwise stated, **there is no particular task (or extrinsic reward**), thereby the *tree-policy* is uniform. The observations of the agent are its coordinates; its actions are the movements into the four cardinal directions. Our hyperparameters can be found in Appendix C.o maximize the entropy of the intra-skill policy with a discrete action space, we use the DQN algorithm [30]. The agent starts an episode at the position of the arrow (see figures) and an episode resets every 100 steps, thus the intra-skill policy lasts 100 steps. At the end of the training phase, the skills of all the nodes are evaluated through an evaluation phase lasting 500 steps for each skill. In all figures, each tile corresponds to a skill with $|V| = 4$ that is displayed at the top-left of the tile. Figure 2 and 4 display the density of the states visited by intra-skill policies during the evaluation phase: the more red the state, the more the agent goes over it.

Do the split of skills improve the exploration of an agent ? Figure 2 shows some skills learned in an environment of 4 rooms separated by a bottleneck. The full set of skills is displayed in Appendix F.e first notice that the agent clearly separates its first skills $(0), (1), (2), (3)$ since the states covered by one skill are distinct from the states of the other skills. However it does not escape from its starting room when it learns these first skills. When it develops the skills close to bottlenecks, it learns to go beyond and invests new rooms. It is clear for skills (1) and (2) which, with one refinement, respectively explore the top-right (skill $(1, 0)$) and bottom-left (skills $(2, 0), (2, 2), (2, 3)$) rooms. With a second refinement, $(2, 3, 0)$ even manages to reach the farthest room (bottom-right). This stresses out that the refinement of a skill also allows to expand the states covered by the skill, and thus can improve the exploration of an agent when the rewards are sparse in an environment.

Do the split of skills correct a wrong over-generalization of a parent skill ? Figure 3 shows the evolution of the intrinsic reward function for some skills (see Appendix G for the full set of skills) The environment contains a vertical wall and settings are the same as before, except the Boltzmann parameter set to 0.5. At the beginning, skill (1) is rewarding identically left and right sides of the wall. This is due to the generalization over coordinates and to the fact that the agent has not yet visited the right side of the wall. However it is a wrong generalization because left and right sides are not close to each other (considering actions). After training, when the agent begins to reach the right side through the skill $(3, 2)$, it corrects this wrong generalization. The reward functions better capture the distance (in actions) between two states: states on the right side of the wall are attributed to skill (3) rather than (1). We can note that other parts of the reward function remain identical.

Can the agent choose which skill to develop as a priority

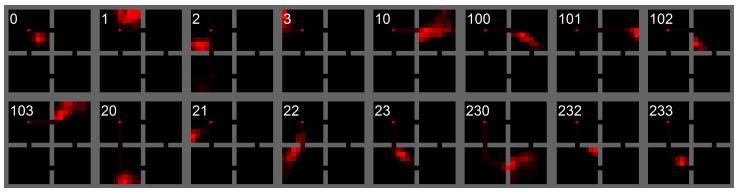


Figure 2: Some skills learned by the agent in an environment composed of four rooms.

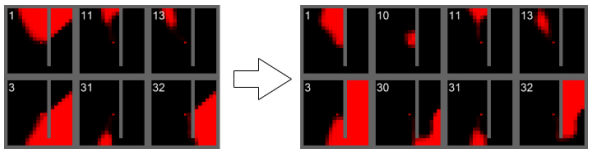


Figure 3: Discriminator's probability of achieving skills (1), (3) and their sub-skills in every state (*i.e.* $q(g|s)$). The more red the state, the more rewarding it is for the skill. The left side corresponds to the preliminary stage of the learning process (timestep 128.10^4); the right side corresponds to the end of the learning process (timestep 640.10^4).

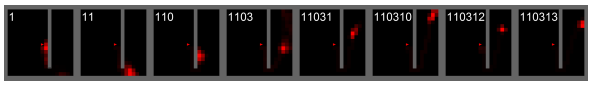


Figure 4: One path in the tree of skills learned by the agent with an extrinsic reward of 1 on the upper right side of the wall.

? In this part, we use the same environment as previously, but states on the right side of the wall give an extrinsic reward of 1. Thus the agent follows the *tree-policy* to maximize its rewards, using Boltzmann exploration, and focus its refinement on rewarding skills. Figure 4 shows all the parent skills of the most refined skill which reaches $L(g) = 6$. The agent learns more specialized skills in the rewarding area than when no reward is provided (cf. Appendix H for the full set of skills learned)

Summary. We illustrated the following properties of ELSIM: 1- it expands a previously learned rewarding area when it discovers new states; we show in Section 4.2 that it improves exploration when the rewards are sparse; 2- adding letters corrects over-generalization of their parent discriminator; 3- it can focus the skill expansion towards task-interesting areas.

4.2 Performance on a single task

In this part, we study the ability of ELSIM to be competitive on high-dimensional benchmarks [1] **without any prior knowledge**. Unless stated otherwise, used hyper-parameters can be found in Appendix D.or ELSIM, we set the maximum skill length to 10, which is reached in *HalfCheetah*.

Figure 5 respectively shows the average reward per episode for different environments. Shaded areas color are upper-bounded (resp. lower-bounded) by the maximal (resp. minimal) average reward.

First, the *MountainCarContinuous* environment represents a challenge for the exploration as it is a sparse reward environment: the agent receives the reward only when it reaches the goal. In this environment, ELSIM outperforms SAC by getting a higher average reward. It confirms our results (cf. Section 4.1) on **the positive impact of ELSIM on the exploration**. There is a slight decrease after reaching an optima, in fact, ELSIM keeps discovering skills after finding its optimal skill. On *Pendulum* and *LunarLander*, ELSIM achieves the same asymptotic average reward than SAC, even though ELSIM may require more timesteps. On *HalfCheetah*, SAC is on average better than ELSIM. However we emphasize that **ELSIM also learns other skills**. For example in *HalfCheetah*, ELSIM learns to walk and flip while SAC, that is a non-hierarchical algorithm, only learns to sprint.

4.3 Transfer learning

In this section, we evaluate the interest of ELSIM for transfer learning. We take skills learned by intra-policies in Section 4.2, reset the *tree-policy* and restart the learning process on *HalfCheetah* and *HalfCheetah-Walk*. *HalfCheetah-Walk* is a slight modification of *HalfCheetah* which makes the agent target a speed of 2 (cf. Appendix J for more details on the new reward function) Intra-skill policies learning was stopped in *HalfCheetah*.

The same parameters as before are used, but we use MBIE-EB [40] to explore the tree (cf. Appendix I for details) Figure 6 shows that the *tree-policy* learns to reuse its previously learned skills on *HalfCheetah* since it almost achieves the same average reward as in Figure 5. On *HalfCheetah-Walk*, we clearly see that the agent has already learned skills to walk and that it easily retrieves them. In both environments, ELSIM learns faster than SAC, which learn from scratch. It demonstrates that skills learned by ELSIM can be used for **other tasks** than the one it has originally been trained on.

5 Related work

Intrinsic motivation in RL is mostly used to improve exploration when rewards are sparse [10] or to learn skills [4].

[1] *HalfCheetah* has a state space and action space respectively of 17 and 6 dimensions.

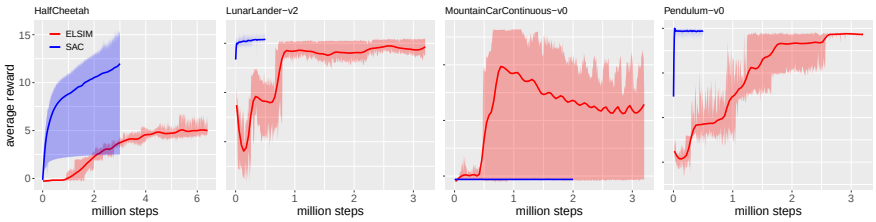


Figure 5: Average reward per episode in classical environments (*HalfCheetah-v2* [46], *LunarLanderContinuous-v0* [38], *MountainCarContinuous-v0* [31] and *Pendulum-v0*) for SAC and ELSIM (averaged over 4 seeds). We use our own implementation of SAC except for *HalfCheetah* for which the blue curve is the average reward of SAC on 5 seeds, taken from [20]. We stopped the simulation after convergence of SAC.

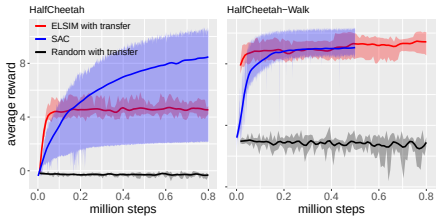


Figure 6: Average reward per episode in *HalfCheetah* and *HalfCheetah-Walk*. We use our own implementation of SAC for *HalfCheetah-Walk*. The black curve is the average reward of a random *tree-policy* that uses the transfered skills.

The works that learn skills with intrinsic rewards are close to our approach and can be classified in two major categories. The first category strives to explicitly target states. The reward is defined either as a distance between agent's state and its goal state [23], or as the difference between the change in the state space and the required change [33]. However, to efficiently guide the agent, the reward functions require a good state representation [35, 32].

Intrinsic motivation as diversity heuristic. Our work mostly falls into this category, which strives to define a skill based on a MI objective (cf. Section 2.2). Seminal works already learn a discrete set of diverse skills [16, 14]. In contrast to them, we manage to learn both the skill and the skill-selection policy in an end-to-end way and we propose an efficient way to learn a large number of skills useful for the tasks the agent wants to accomplish. Recent works [48, 12] try to learn a continuous embedding of skills, but do not integrate their work into an end-to-end hierarchical learning agent. DADS [39] learn skills using a generative model over observations rather than over skills. While this is efficient on environments with simple observation space, this is computationally ineffective. In our work, rather than learning a continuous skill embedding, we strive to select and learn skills among a very large number of discretized skills. As a consequence, we focus our learned skill distribution only on task-interesting skills and we do not rely on a parametric state distribution.

Continual learning. Other works proposed a *lifelong learning* architecture. Some assume that skills are already learned and learn to reuse them; for example H-DRLN [44] uses a hierarchical policy to choose between ground actions and skills. They also propose to distill previously learned skills into a larger architecture, making their approach scalable. In contrast, we tackle the problem of learning skills in an end-to-end fashion, thereby our approach may be compatible. Similarly to us, CCSA [22] addresses the catastrophic forgetting problem by freezing the learning of some experts. They mix two unsupervised learning methods to find and represent goal states, and then learn to reach them. However, their unsupervised algorithm only extracts linear features and they manually define a first set of skills. One particular aspect of continual learning is Meta-RL: how can an agent learn how to learn ? Traditional methods assume there exists a task distributions and try to generalize over it [15, 13]; this task distribution serves as prior knowledge. In [19], the authors address this issue and apply MAML [15] on an uniform distribution of tasks learned by DIAYN [14]. However, learning is neither focused, nor end-to-end. In the continuity of this work, CARML [21] mixes the objective of DADS [39] and Meta-RL; it alternates between generating trajectories of the distribution of tasks and fitting the task distribution to new trajectories. While CARML discovers diverse behaviors with pixel-level state space, it cannot learn a global objective end-to-end like ELSIM.

State abstraction. Our method can be viewed as a way to perform state abstraction [24]. Rather than using this abstraction as inputs to make learning easier, we use it to target specific states. The application of our refinement method bounds the suboptimality of the representation, while the task-independent clustering ensures that skills are transferable. In contrast to our objective, existing methods usually tackle suboptimality for a task without addressing transfer learning or exploration [3, 1]. The k-d tree algorithm [17] has been used to perform state abstraction over a continuous state space [47], but as above, the splitting process takes advantage of extrinsic reward and previously defined partitions are not adapted throughout the learning process. In the domain of developmental robotics, RIAC and SAGG-RIAC [6, 7] already implement a splitting al-

gorithm building a tree of subregions in order to efficiently explore the environment and learn a forward model. More precisely, they split the state space to maximize either the sum of variance of interactions already collected or the difference of learning progress between subregions. However, these heuristics do not scale to larger continuous environments. In contrast, we assign states to subregions according to the proximity of states and use these subregions as reusable skills to solve several tasks. ASAP [28] partitions the goal space, but does not use intrinsic motivation and the partitions are limited to hyper-plans.

6 Conclusion

We proposed ELSIM, a novel algorithm that continually refines discrete skills using a recently defined diversity heuristic [14]. To do so, the agent progressively builds a tree of different skills in the direction of a high-level objective. As shown in Section 4, **ELSIM expands the area associated to a skill** thanks to its exploratory behavior which comes from adding latent variables to the overall policy. **ELSIM also focuses its training on interesting skills relatively to some tasks**. Even though the agent is often learning a task, the skills can be defined independently from a specific task and we showed that ELSIM possibly makes them **transferable across different tasks** of a similar environment. Since the agent does not need extrinsic reward to learn, we show that it can **improve exploration** on sparse rewards environments. We believe that such a paradigm is appropriate for *lifelong learning*.

Currently, our method allows to avoid the problem of catastrophic forgetting, but the counterpart is an increase of the memory footprint, which is a recurrent issue in methods based on trees. Several works addressing catastrophic forgetting may be adapted to our work, e.g. [25] and could potentially improve transfer learning between neural networks at different levels of our tree. In addition, ELSIM quickly gets stuck in local optimas in more difficult environments such as *BipedalWalker-v2* or Pybullet environments. The main limitation of our approach is that we cannot select several skills in one episode, such as one would make within the *option* framework [42]. To be adapted, the *tree-policy* should be dependent on the true state and the diversity heuristic should maximize $\mathbb{E}_{s\sim\mathbb{U}(s)}\, I(G, S'|S)$ rather than Equation (6) like in [39]. Thus the curriculum algorithm should be modified. It would result that the semantic meaning of a skill would be no longer to target an area, but to produce a change in the state space. We plan to address these issues in future work.

7 Acknowledgment

We thank O. Sigaud and A. Dutech for their useful feedbacks on the paper. We acknowledge the support of NVIDIA Corporation with the donation of the Titan V used for this research.

References

[1] Abel, D., Hershkowitz, D.E., Littman, M.L.: Near optimal behavior via approximate state abstraction. In: ICML. pp. 2915–2923 (2016)

[2] Achiam, J., Edwards, H., Amodei, D., Abbeel, P.: Variational option discovery algorithms. arXiv preprint arXiv:1807.10299 (2018)

[3] Akrour, R., Veiga, F., Peters, J., Neumann, G.: Regularizing reinforcement learning with state abstraction. In: IROS. pp. 534–539. IEEE (2018)

[4] Aubret, A., Matignon, L., Hassas, S.: A survey on intrinsic motivation in reinforcement learning. arXiv preprint arXiv:1908.06976 (2019)

[5] Bacon, P.L., Harb, J., Precup, D.: The option-critic architecture. In: AAAI. pp. 1726–1734 (2017)

[6] Baranes, A., Oudeyer, P.Y.: R-iac: Robust intrinsically motivated exploration and active learning. IEEE Transactions on Autonomous Mental Development **1**(3), 155–169 (2009)

[7] Baranes, A., Oudeyer, P.Y.: Intrinsically motivated goal exploration for active motor learning in robots: A case study. In: IROS. pp. 1766–1773 (2010)

[8] Barber, D., Agakov, F.V.: The im algorithm: a variational approach to information maximization. In: Advances in neural information processing systems. pp. 201–208 (2003)

[9] Barto, A.G.: Intrinsic motivation and reinforcement learning. In: Intrinsically motivated learning in natural and artificial systems, pp. 17–47. Springer (2013)

[10] Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R.: Unifying count-based exploration and intrinsic motivation. In: Advances in Neural Information Processing Systems. pp. 1471–1479 (2016)

[11] Chevalier-Boisvert, M., Willems, L.: Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid` (2018)

[12] Co-Reyes, J.D., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., Levine, S.: Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In: ICML. pp. 1008–1017 (2018)

[13] Duan, Y., Schulman, J., Chen, X., Bartlett, P.L., Sutskever, I., Abbeel, P.: Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. CoRR **abs/1611.02779** (2016)

[14] Eysenbach, B., Gupta, A., Ibarz, J., Levine, S.: Diversity is all you need: Learning skills without a reward function. In: ICLR (2019)

[15] Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 1126–1135. JMLR. org (2017)

[16] Florensa, C., Duan, Y., Abbeel, P.: Stochastic neural networks for hierarchical reinforcement learning. In: ICLR (2017)

[17] Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM (TOMS) **3**(3), 209–226 (1977)

[18] Gregor, K., Rezende, D.J., Wierstra, D.: Variational intrinsic control. In: ICLR 2017 (2017)

[19] Gupta, A., Eysenbach, B., Finn, C., Levine, S.: Unsupervised meta-learning for reinforcement learning. CoRR **abs/1806.04640** (2018)

[20] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: ICML. pp. 1856–1865 (2018)

[21] Jabri, A., Hsu, K., Gupta, A., Eysenbach, B., Levine, S., Finn, C.: Unsupervised curricula for visual meta-reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 10519–10530 (2019)

[22] Kompella, V.R., Stollenga, M., Luciw, M., Schmidhuber, J.: Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. Artificial Intelligence **247**, 313–335 (2017)

[23] Levy, A., Platt, R., Saenko, K.: Hierarchical reinforcement learning with hindsight. In: International Conference on Learning Representations (2019)

[24] Li, L., Walsh, T.J., Littman, M.L.: Towards a unified theory of state abstraction for mdps. In: ISAIM (2006)

[25] Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continual learning. In: Advances in Neural Information Processing Systems. pp. 6467–6476 (2017)

[26] Machado, M.C., Bellemare, M.G., Bowling, M.: A laplacian framework for option discovery in reinforcement learning. In: ICML. vol. 70, pp. 2295–2304. JMLR. org (2017)

[27] Machado, M.C., Bowling, M.: Learning purposeful behaviour in the absence of rewards. arXiv preprint arXiv:1605.07700 (2016)

[28] Mankowitz, D.J., Mann, T.A., Mannor, S.: Adaptive skills adaptive partitions (asap). In: Advances in Neural Information Processing Systems. pp. 1588–1596 (2016)

[29] Metzen, J.H., Kirchner, F.: Incremental learning of skill collections based on intrinsic motivation. Frontiers in neurorobotics **7**, 11 (2013)

[30] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)

[31] Moore, A.W.: Efficient memory-based learning for robot control (1990)

[32] Nachum, O., Gu, S., Lee, H., Levine, S.: Near-optimal representation learning for hierarchical reinforcement learning. In: ICLR (2019)

[33] Nachum, O., Gu, S.S., Lee, H., Levine, S.: Data-efficient hierarchical reinforcement learning. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 3303–3313 (2018)

[34] Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., Levine, S.: Why does hierarchy (sometimes) work so well in reinforcement learning? arXiv preprint arXiv:1909.10618 (2019)

[35] Nair, A.V., Pong, V., Dalal, M., Bahl, S., Lin, S., Levine, S.: Visual reinforcement learning with imagined goals. In: Advances in Neural Information Processing Systems. pp. 9209–9220 (2018)

[36] Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)

[37] Ryan, R.M., Deci, E.L.: Intrinsic and extrinsic motivations: Classic definitions and new directions. Contemporary educational psychology **25**(1), 54–67 (2000)

[38] Shariff, R., Dick, T.: Lunar lander: A continous-action case study for policy-gradient actor-critic algorithms (2013)

[39] Sharma, A., Gu, S., Levine, S., Kumar, V., Hausman, K.: Dynamics-aware unsupervised discovery of skills. arXiv preprint arXiv:1907.01657 (2019)

[40] Strehl, A.L., Littman, M.L.: An analysis of model-based interval estimation for markov decision processes. Journal of Computer and System Sciences **74**(8), 1309–1331 (2008)

[41] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)

[42] Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence **112**(1-2), 181–211 (1999)

[43] Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research **10**(Jul), 1633–1685 (2009)

[44] Tessler, C., Givony, S., Zahavy, T., Mankowitz, D.J., Mannor, S.: A deep hierarchical approach to lifelong learning in minecraft. In: AAAI (2017)

[45] Thrun, S.: Is learning the n-th thing any easier than learning the first? In: Advances in neural information processing systems. pp. 640–646 (1996)

[46] Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: IEEE/RSJ IROS. pp. 5026–5033. IEEE (2012)

[47] Uther, W.T., Veloso, M.M.: Tree based discretization for continuous state space reinforcement learning. In: Aaai/iaai. pp. 769–774 (1998)

[48] Warde-Farley, D., de Wiele, T.V., Kulkarni, T.D., Ionescu, C., Hansen, S., Mnih, V.: Unsupervised control through non-parametric discriminative rewards. In: ICLR (2019)

A *Tree-policy* algorithm

Algorithm 1 shows how ELSIM runs an episode in the environment without the learning part of the intra-skill policy and the discriminator. There are 3 steps: 1-an agent runs an episode inside the MDP of skills; the sequence of actions represents a skill; 2- the agent executes the intra-skill policy of the skill; 3- the *tree-policy* is rewarded according to how well the intra-skill policy fits the task and the Q-learning applies.

Algorithm 1 *Tree-policy* of ELSIM

Require: Environment env, episode length ep_{len}, learning rate ϕ.
Require: Tree T, *Tree-policy* π_T.
Require: Leaves' policies and buffers: $\forall g_{leaves} \in leaves(T)$, $\pi_\theta^{g_{leaves}}$, $\mathbb{B}^{g_{leaves}}$.
{It selects a skill to execute}
$node \leftarrow root(T)$
while $not(leaf(node))$ **do**
  $node \leftarrow Boltzmann(\pi_T(node))$
end while
$r_{tree} \leftarrow 0$
{It runs the intra-skill policy in the environment}
$obs \leftarrow env.reset()$
$done \leftarrow false$
while $not(done)$ **do**
  $action \leftarrow \pi^{node}(obs)$
  $next\_obs$, $reward$, $done \leftarrow env.step(action)$
  Fill $\mathbb{B}^{node}$ with $obs, done, next\_obs, action$
  $obs \leftarrow next\_obs$
  $r_{tree} \leftarrow r_{tree} + \gamma_T \frac{reward}{ep_{len}}$
end while
{It learns Q-values of the *tree-policy*}
$parent \leftarrow parent(node)$
$Q(parent, node) = (1 - lr_T) \times Q(parent, node) + lr_T \times r_{tree}$
while $not(root(node))$ **do**
  $parent \leftarrow parent(node)$
  $Q(parent, node) = \max_{n \in children(node)}(Q(node, n))$
end while

B Learning algorithm

Algorithm 2 shows one learning step in ELSIM for discriminators and intra-skill policies. The agent executes its *tree-policy* on its tree; the discriminators of the intermediary encountered nodes learn with probability η on a batch of interactions of their recursively and uniformly chosen children leaves. The last discriminator learns with probability 1 over a batch of its leaves' interactions. This batch is labeled with the intrinsic reward computed through Equation (6) and leaves' intra-skill policies learn with this batch.

Algorithm 2 Learning step of intra-skill policies and discriminators

Require: $batch\_size$, Tree T, *Tree-policy* π_T.
Require: Discriminators: $\forall g \in nodes(T),\ q_\omega^g$.
Require: Leaves' policies and buffers: $\forall g_{leaves} \in leaves(T),\ \pi_\theta^{g_{leaves}},\ \mathbb{B}^{g_{leaves}}$.
{It selects a node to learn on}
$node \leftarrow root(T)$
while $not(leaves(children(node)))$ **do**
  if $random() < \eta$ **then**
    $i \leftarrow 0$ {Discriminators in exploitation phase learn with probability η}
    $batch \leftarrow \emptyset$
    while $i < batch\_size$ **do**
      $node2 \leftarrow node$
      while $not(leaf(node2))$ **do**
        $node2 \leftarrow Uniform(children(node2))$
      end while
      $ADD(batch, sample(\mathbb{B}^{node2}))$
    end while
    $Cross\_entropy(batch, q_\omega^{node})$
  end if
  $node \leftarrow Boltzmann(\pi_T(node))$
end while
{Learn the discriminator and intra-skill policies of the last node}
$i \leftarrow 0$
$batch \leftarrow \emptyset$
while $i < batch\_size$ **do**
  $leaf \leftarrow Uniform(children(node))$
  $ADD(batch, sample(\mathbb{B}^{leaf}))$
end while
$Cross\_entropy(batch, q_\omega^{node})$
for $leaf \in children(node)$ **do**
  Learn π_θ^{leaf} with Equation (6) and SAC
end for

C Hyper-parameters on gridworlds

Table 1 shows hyperparameters used in gridworlds experiments. Hyper-parameters of the discriminator and DRL networks are identical. *Tree-policy* parameters are used only when there is a high-level goal.

Parameters	Symbol	Values
DRL		
Boltzmann coefficient	α_{DQN}	1
Buffer size	$\mathbb{B}_{size}$	10k
Hidden layers	hl_{SAC}	2x64
Learning rate	lr_{DQN}	0.001
Gamma	γ	0.98
Episode duration	D	100
Batch size	$batch\_size$	64
Parallel environments	n	16
target smoothing coefficient	τ	0.005
ELSIM		
Discriminators hidden layers	lr_D	2x64
Split threshold	δ	0.9
Sampling probability	η	0.5
Average coefficient	β	0.02
Vocabulary size	$\|V\|$	4
Old discriminations scale	α	1
Tree-policy		
Gamma	γ_T	1
Boltzmann coefficient	α_T	20
Learning rate	lr_T	0.05

Table 1: Hyper-parameters used for gridworld experiments.

D Hyper-parameters on continuous environments.

Since the goal of ELSIM is to learn in a continual learning setting, we record the performances of ELSIM in training mode, *i.e.* with its stochastic policies. Table D shows hyper-parameters used in experiments on continuous environments. Learning rate of the discriminator and DRL networks are identical. In addition to these hyper-parameters, we set the weight decay of discriminators to 0.01 in the exploitation phase. We also manually scale the intra-skill rewards and entropy coefficient α_{SAC} to get lower-magnitude value function, thus, we divide rewards and entropy coefficient by 5 and clamp the minimal reward by (-2). SAC algorithms used by ELSIM do not use a second critic (but our implementation of SAC does).

Classic environments . On MountainCar, Pendulum and LunarLander , we changed α_{sac} to 0.1.

Our implementation of SAC . Our implementation of SAC use the same hyper-parameters as in [20], but with a buffer size of 100000 and a learning rate of 0.001.

Transfer learning . In experiments on transfer learning, we fix the number of parallel environments to 1 and reduce the length of an episode to 200. For the first 20 updates of a node, the *tree-policy* remains uniform.

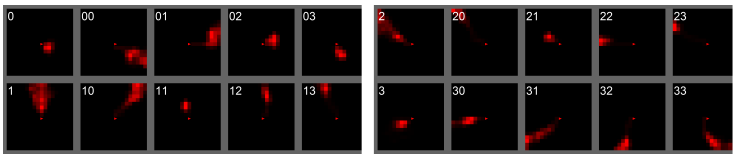


Figure 7: Different states covered by the agent while doing a skill. The first and sixth columns display the intra-skill policies learned with a message length equal to 1; once the learning has been completed, the agent refines each skill into four new sub-skills, displayed on each row.

Parameters	Symbol	Values
DRL		
Entropy coefficient	α_{SAC}	0.25
Buffer size	$\mathbb{B}_{size}$	20k
SAC hidden layers	hl_{SAC}	2x128
Learning rate	lr_{SAC}	0.001
Discount	γ	0.98
Episode duration	D	500
Batch size	$batch\_size$	128
Parallel environments	n	16
target update parameter	τ	0.005
ELSIM		
Discriminators hidden layers	lr_D	2x64
Split threshold	δ	0.9
Sampling probability	η	0.5
Average coefficient	β	0.02
Vocabulary size	$\|V\|$	4
Old discriminations scale	α	2
Tree-policy		
Gamma	γ_T	1
Boltzmann coefficient	α_T	5
Learning rate	lr_T	0.05

Table 2: Hyper-parameters used for experiments on continuous environments.

E Refining a high-stochastic skill into several low-stochastic skills

The first column of Figure 7 shows, for each possible skill g when $L(g) = 1$, the states covered by the agent during the evaluation phase of the intra-skill policies. We can see that the agent clearly separates its skills since the states covered by one skill are distinct from the states of the other skill. In our example, the first skill (0) makes the agent go at the right of the grid, the second one (1) at the top, the third one (2) at the left and the fourth one at the bottom. In contrast, columns 2-5 of Figure 7 show the intra-skill policies learned with $L(g) = 2$. As evidenced by goals' numbers, the fours rightmost intra-skill policies are the refinement of the leftmost fuzzy policy on the same row. We see that the refinement allows to get lower-stochastic policies. For example, skill (1) is very fuzzy while its children target very specific areas of the world. This emphasizes the benefits of using more latent variables to control the environment.

F Skills learned in four rooms environment

Figure 8 shows the complete set of learned skills in four rooms environment. It completes skills displayed by Figure 2.

G Skills learned with a vertical wall

Figure 9 shows the evolution of the reward function for the complete set of learned skills. It completes skills displayed by Figure 3.

H Skill expansion

Figure 10 shows the complete set of learned skills learned in an environment with a vertical wall. States on the upper right side of the wall give a reward of 1. It completes skills displayed by Figure 4.
In Figure 11, we perform the same simulation as in Figure 4 without goals. The *tree-policy* does not focus on the right side of the wall, and thus, gets less controllability than in Figure 4.

I Exploration strategy of the *tree-policy* with transfer learning

To quickly learn to use skills despite their duration, we used MBIE-EB [40] exploration strategy. This method adds a count-based bonus to the Q-value:

$$\tilde{Q}(s,a) = Q(s,a) + \frac{\beta}{\sqrt{n(s,a)}} \tag{9}$$

where $n(s,a)$ is the number of time we chose a in s and β is an hyper-parameter. We set it to 10 for classic HalfCheetah and 2 for walking HalfCheetah. The agent selects the action that has the larger $\tilde{Q}$.

J *HalfCheetah-Walk*

We introduce a slight modification of *HalfCheetah-v2* to make the creature walk. The reward function used in *HalfCheetah-v2*:

$$R(s,a,s') = Sf(s,s') + C(a) \tag{10}$$

where $C(a)$ is the cost for making moves and $Sf(s,s')$ is the speed of the agent. The reward function used in *HalfCheetah-Walk* is:

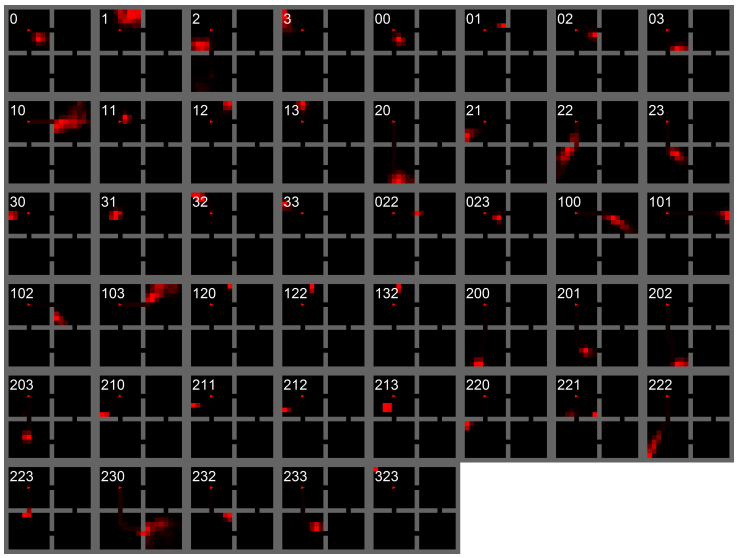


Figure 8: Full set of skills learned by the agent in an environment composed of four rooms.

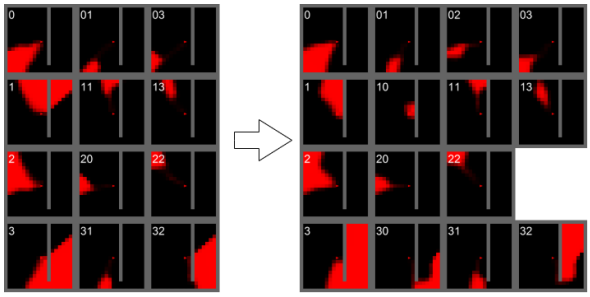


Figure 9: Probability of achieving each skill in every states (*i.e.* $q(g|s)$).

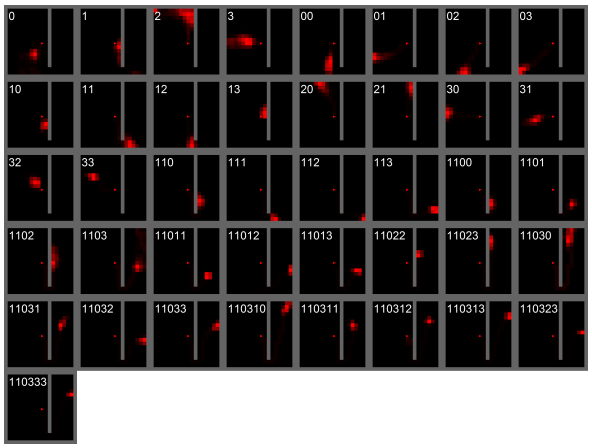


Figure 10: Skills learned by the agent in an environment with a vertical wall.

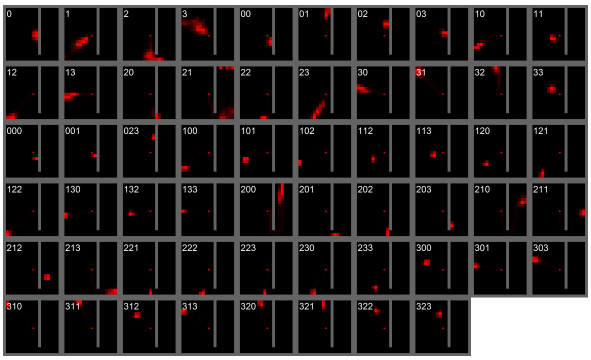


Figure 11: Skills learned by the agent in an environment with a vertical wall. The agent does not focus on a specific area since there are no rewarding states.

$$R(s,a,s') = \begin{cases} Sf(s,s') + C(a) & if\ Sf(s,s') > 2 \\ 4 - Sf(s,s') + C(a) & else \end{cases} \quad (11)$$

Accélération de la simulation d'*Emulatio*, un jumeau numérique de schéma électrique par fusion de données et intelligence augmentée

Dr. Jérémy PATRIX
Responsable R&D
Opal Research[†]

Dr. Beranger SIX
CRIStAL - UMR 9189[*]
Ccamy Systèmes

Sylvain LINTZ
Directeur de la recherche
Ccamy Systèmes[‡]

Résumé

Notre action tente de résoudre les problématiques posées par l'émulation temps-réel des systèmes de contrôle-commande embarqués (C&C) de projets ferroviaires. L'objectif est d'accélérer le cycle de validation des architectures électriques en expérimentant des scénarios sur leur jumeau numérique, afin de supprimer les erreurs de conception qui engendrent dangers, retards et surcoûts. Notre simulateur est exécuté en TrainLab (Laboratoire industriel d'essais ferroviaires basé sur un modèle Hardware-in-the-Loop) pour validation des interfaces électriques des équipements de C&C et de surveillance du matériel roulant. En intégrant fusion de données et intelligence augmentée, nous accélérons Emulatio pour atteindre des temps quasi-réels, tout en maintenant les performances de fiabilité des systèmes électriques.

Mots-clefs

Agents autonomes et systèmes multi-agents : simulation, Systèmes à base de règles, Aide à la décision, Fouille de données, Ordonnancement, Raisonnement à base de modèles, Programmation par contraintes, Extraction et gestion des connaissances, Émulation temps-réel en conception électronique, Jumeau numérique de schéma électrique, Simulation Hardware-in-the-Loop.

Abstract

Our action attempts to solve the problematics raised by the real-time emulation of on-board control-command (C&C) systems for railway projects. The goal is to speed-up the validation cycle of electrical architectures by experimenting scenarios on their digital twin, in order to eliminate design errors generating threats, delays and additional costs. Our simulator is run in TrainLab (Industrial Laboratory for Railway Testing based on a Hardware-in-the-Loop model) for validation of electrical interfaces of C&C and rolling stock monitoring equipment. By integrating data fusion and enhanced intelligence, we accelerate Emulatio to reach near-real time, while maintaining the reliability performance of electrical systems.

[*]CRIStAL : **C**entre de **R**echerche en **I**nformatique, **S**ignal **e**t **A**utomatique de **L**ille

[†]OPAL : Institut de recherche privé en **O**rganisation de la **P**roduction, **A**ménagement du Territoire et **L**ogistique, avec des compétences en socio-économie des transports et des problématiques de *supply chain*.

[‡]Ccamy Systèmes a rejoint Vulcain Ingénierie en 2019.

Keywords

Autonomous agents and multi-agent systems : simulation, Rules-based systems, Decision-aid, Data mining, Ordering, Reasoning based on models, Constraint programming, Knowledge extraction and management, Real-time emulation in electronics design, Digital wiring diagram twins, Hardware-in-the-loop simulation.

1 Introduction

Dans le monde de l'Embarqué, les interactions sur les schémas (électriques, hydrauliques, pneumatiques) des systèmes de contrôle-commande (C&C) sont souvent représentées de façon simplifiées, dissimulant des erreurs fonctionnelles ou organiques qui seront découvertes lors des phases d'essais industriels. Les constructeurs aéronautiques, ferroviaires, sous-mariniers et automobiles ont alors intégré la simulation dans leur phase de développement projet, afin de réduire les ruptures et d'éviter les réitérations entre les cycles de conception et de validation. Les dates limites inhérentes à ces types de projet obligent à suivre une productivité maximale à moindre coût. Cela est possible **par la validation des schémas, en simulant leurs jumeaux numériques** (avant les étapes de banc d'essais et de mise en production du prototype).

FIGURE 1 – Quelques exemples de projets dont nous avons simulé les systèmes C&C : les surnommés *ICNG* et *TGV2020* (2019-...) ; *RER NG* d'Alstom-Bombardier (2018-2020) ; train *Avelia Liberty* d'Amtrak (2017-2019) ; gamme *tramway Citadis X05* d'Alstom (2015-2025) ; métros de Lille, de Singapour (ligne circulaire) et de Los Teques (Venezuela) ; locomotive au Kazakhstan ; etc.

En *TrainLab*, et peut-être chez tous les constructeurs, le manque de rapidité d'un **jumeau numérique** est maintenant perçu comme un **rouage freinant la chaîne logistique**. Ce frein est causé par une **complexité croissante des schémas électriques** issue de chaque innovation à intégrer et norme à respecter. Par exemple, de 2017 à 2019 (cf. Figure 1), nous avons simulés des schémas électriques passant de 4k à 28k composants et éléments électriques interconnectés par des signaux se croisant à des nœuds (ce qui représentent des graphes non orientés de 7k arêtes reliant 2k sommets devenus 48k arêtes reliant 14k sommets). "Interconnexion" plutôt que "connexion" puisque les composants peuvent être reliés par un nœud virtuel de C&C ou par un nœud réel propageant la tension.

Cet article s'inscrit dans une démarche de recherche-développement (R&D) engagée par *Ccamy Systèmes* pour détecter les "erreurs de conception" [22, 23, 24]. En 2013, nous avons ainsi créé ***Simulatio*** qui depuis a été mis en œuvre en *TrainLab*, initialement sur console de conception puis sur console de **co-simulation**, sur plus de **20 projets industriels d'envergure** des domaines ferroviaires (cf. Figure 1), mais aussi défenses et agricoles.

Nous présentons dans cette introduction le contexte industriel nous ayant amené à créer le projet ***Emulatio*** pour réduire les processus énergivores de validation de schéma électrique ferroviaire de sa conception jusqu'à son prototypage en *TrainLab*. Après les travaux antérieurs d'analyse de circuit électrique pour les simuler, nous présentons nos approches d'intelligence augmentée et de fusion de données, intégrées dans *Emulatio* afin d'obtenir un **émulateur temps-réel** de systèmes C&C (une forme de jumeau numérique), dont la problématique principale est l'accélération des temps de réponse. Nous terminons par les performances mesurées au cours d'expérimentations confirmées chez différents *TrainLabs*.

1.1 La validation énergivore de la CAO

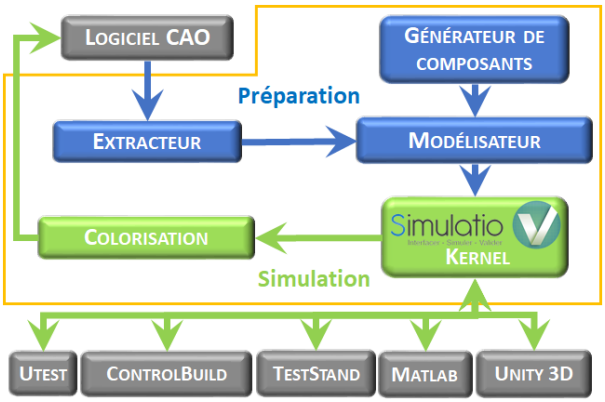


FIGURE 2 – Le pipeline des traitements des données (cf. Figure 6) pour simuler : une phase de préparation et une phase de multiples simulations "colorisant" la CAO (conception assisté par ordinateur). Chaque commande simulée est reçue par un des logiciels propriétaires (gris) et les changements d'état des contrôles leur sont retournés.

La validation est l'activité la plus énergivore des schématistes qui corrigent itérativement chaque version du schéma électrique de sa conception jusqu'au prototype en *TrainLab* (avant son industrialisation). Les étapes de validation sont ordonnancées sur la Figure 2 par : la CAO du schéma électrique, l'extraction des données, la modélisation du graphe de composants, la mise en simulation du *Kernel* qui "colorise" la CAO pendant l'exécution de scénarios de C&C (enregistrés ou commandés par le banc de test). Les schématistes diagnostiquent ainsi visuellement les erreurs de conception (cf. les exemples réels des Figures 3 et 5). Le cycle recommence à chaque correction de la CAO et pour chaque nouvelle version du projet (lorsque les experts rajoutent de nouveaux circuits après obtention des spécifications fonctionnelles à respecter). Ces étapes forment un pipeline des traitements des données que nos travaux R&D optimisent [22, 23, 24] afin d'en réduire les coûts et la complexité.

1.2 La "colorisation" de la CAO

Le terme "colorisation" signifie la mise en couleur de films en noir et blanc, par un procédé électronique. Dans notre application, il s'agit de la mise en couleur des signaux selon chaque tension et des composants C&C selon leurs états. Comme dans les Figures 3 et 5, nous utilisons plusieurs couleurs pour nos projets ferroviaires (train, tram, métro, etc.) :

- Rouge < 10 Volts et Vert pour la gamme des tensions de 12V, 24V, 48V, 72V et 110V ;
- Orange < 0 Volt (cas du potentiel flottant) et Violet > 140V (230V et 400V pour les niveaux de tension "réseau", et 750 V, 1500 V, 3 kV et 25 kV pour les "caténaires") ;
- Bleu pour les parties du schéma non simulées (mais démontrant la possibilité) pour plusieurs raisons possibles : vérifier la sûreté / fiabilité en cas de dysfonctionnement de ces parties, en attente des spécifications fonctionnelles, etc.

Depuis ce résultat, nous pensons que **tout système de C&C interopérable avec du matériel électromécanique pourrait théoriquement être remplacé par le jumeau numérique de son architecture électrique**. Cela permettrait de profiter à la fois des avantages du monde logiciel (maintenance, mise à jour, diagnostic, gestion à distance) et du monde électrique (qui a un niveau de sécurité maximal tels que *SIL4* selon la norme *EN 50657* de sûreté de fonctionnement logiciel ferroviaire).

Emulatio
Diagnostic – Contrôle - Modèle

Le projet de recherche ***Emulatio*** a été lancé pour dépasser les verrous technologiques afin d'**atteindre l'émulation** : cela consiste à substituer un matériel par un logiciel qui imite son comportement physique, et communique en temps réel l'effet de chaque commande (tel que les modifications d'état des contrôles), tout en respectant la totalité des cohérences analogiques des tensions des signaux traversant tous les types d'éléments électriques.

.

FIGURE 3 – Animation de "colorisation" : utilisation de C&C du Pupitre (à droite), lançant des calculs du *Kernel* (*Simulatio* 2016 à gauche) dont les résultats colorisent le logiciel CAO (au centre, en utilisant son *API*).

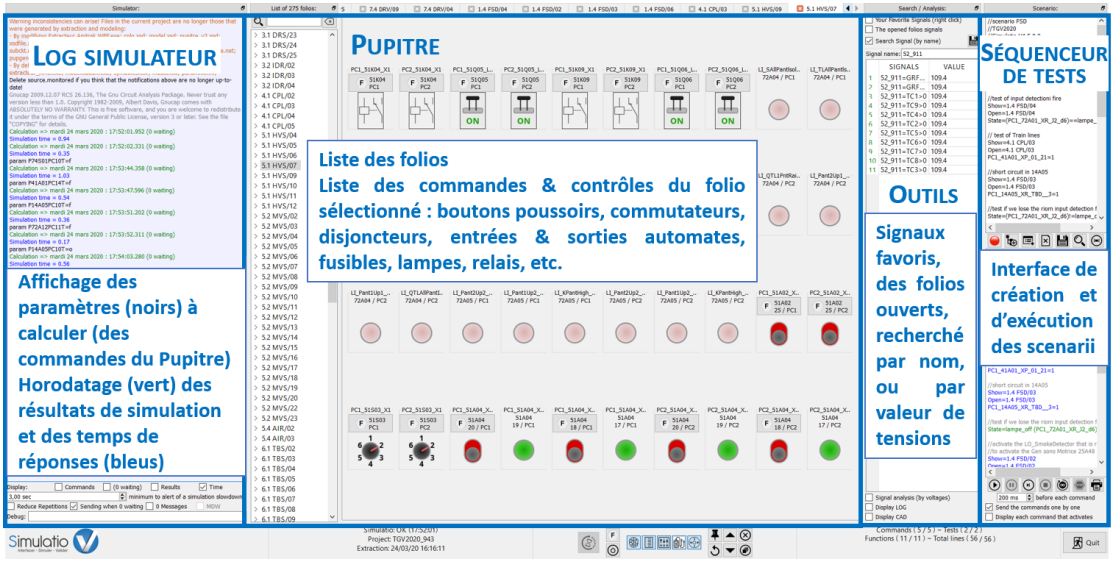


FIGURE 4 – L'IHM de *Simulatio* 2019 (de gauche à droite les modules : Simulateur, Folios, Pupitre, Outils d'analyse des signaux et Scénario, déportable selon le besoin). Vous pouvez observer l'évolution de l'IHM depuis la Figure 3.

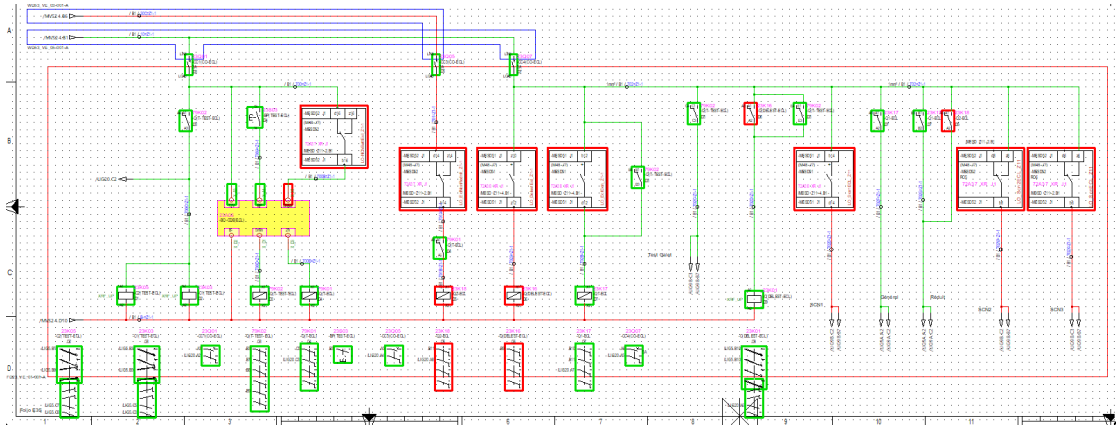


FIGURE 5 – *Simulatio* colorisant E3.series (logiciel CAO de Zuken). **Nos simulations en temps réel sont plus démonstratives.** Les composants sont schématisés avec des *lumped elements models*. Les nœuds y sont représentés par des points bleus d'intersection connectant les composants électriques noirs (avec paramètres, références, configuration à l'état initial, etc.).

2 Travaux antérieurs

Notre expérience technologique des simulateurs nous a montré que, pour un même nombre de composants dans deux circuits équivalents, les simulateurs sont plus lents sur le circuit ayant les interconnexions les plus complexes. Leurs algorithmes sont plus performants en fonction de la modélisation des données. Nous avons effectué un état de l'art sur la fusion de données et l'intelligence artificielle. Nous ne pouvons pas tenter le *deep learning* puisque dans les simulateurs logiques (à seuils), les incertitudes (faux positifs / négatifs) se produisent justement dans de nombreux **cas analogiques indéterminables logiquement** : bagotement [1], auto-maintien, court-circuit, ... donc toute interaction et tension simulée sans prise en compte des phénomènes physiques. Pour la fusion de données, en génie électrique, le thème le plus proche est le *model order reduction* se basant sur l'analyse de circuit.

2.1 Analyse de circuit électrique

L'analyse de circuit (*network analysis*) est le processus consistant à trouver les tensions passant à travers chaque élément du réseau. Pour calculer ces valeurs, les composants du réseau sont supposés linéaires, et est posé un système d'équations à résoudre (dans lequel les inconnues seront les intensités et certaines tensions) en se basant sur :

- La **méthode / loi des nœuds** (*nodal-voltage analysis* ou *branch current method*) de *Kirchhoff* : "la somme des courants entrant est égale à la somme de ceux sortant".

- La **méthode / loi des mailles** (*mesh analysis* ou *mesh current method*) de *Kirchhoff* : "la somme des tensions est nulle dans un réseau" constitué de cycles (sans sous-cycle).

Afin d'obtenir les tensions aux nœuds sur l'état courant d'un schéma électrique, le simulateur *Gnucap* (que nous avons intégré, cf. la partie 3) exécute une analyse nodale par itération en se basant sur :

- La **méthode de Newton** détecte les cas de non-convergence lors de la recherche du point de fonctionnement, évitant que le simulateur tourne en boucle lorsque les tensions ne se stabilisent pas (ex : bagotement[1], circuit à composants très nombreux et/ou fortement couplés).

- La **décomposition *LU*** résout un système d'équations non linéaires, étant une forme matricielle d'une élimination *gaussienne*, dont l'inconnue est la tension des nœuds.

Le fonctionnement de *Gnucap* suit le concept qu'une queue d'évènements et une matrice incrémentale se mettent à jour plus rapidement pour les grands circuits. Des simulateurs utilisent des alternatives ayant des avantages, mais surtout leur complexité : décomposition de *Cholesky* / *QR*, élimination du *Gauss-Jordan*, méthode de *Gauss-Seidel* / *SOR* / *Jacobi* / ..., résumé en deux types de simulateurs :

1. Oscillation erratique entre plusieurs valeurs ou états, par exemple avec un relais lorsque le champ magnétique d'une bobine ouvre le contact qui alimentait cette bobine ce qui ferme le contact qui ré-alimente la bobine, et ainsi de suite empêchant de déterminer son état.

- **Analogique** : lent mais évite les incohérences et traite des signaux continus dans le temps tels que les valeurs de tension, impédance, etc.

- **Logique** : plus rapide, manipule des signaux discrétisés et quantifiés tels que 0, 1, *X*, indéterminé.

Ces simulateurs suivent le même processus :

1. Lecture de la description du circuit,
2. Mise en équation optimale,
3. Résolution de ces équations avec la méthode correspondante dans le régime demandé,
4. Retourne le résultat sous forme de tableaux ou de courbes.

Ils ont besoin de données décrivant le schéma électrique et d'au moins deux librairies pour :

- les comportements de chaque type de composants électriques existants, et

- la résolution d'équations (différentielles par exemple), comme *BLAS* (*Basic Linear Algebra Subprograms*) pour effectuer du calcul à haute performance sur trois niveaux de complexité (linéaire, quadratique et cubique). Elle s'adapte à l'architecture hardware et permet du *SIMD* (*Single Instruction on Multiple Data*, une forme de parallélisme sur un seul cœur du processeur).

Comme stratégie, le calcul est ordonné selon le transfert d'énergie (ou d'information). Le fait de produire des équations en exploitant la loi des mailles et celle des nœuds permet d'utiliser directement des (macro-)modèles mathématiques pour répliquer le comportement des circuits entiers du schéma électrique qui s'exécuteront de manière maîtrisée et avec moins d'incohérences. Ces modèles sont issus d'approches de "réduction d'ordre du modèle" permettant à l'analyse de circuit du simulateur d'accélérer par moins d'équations (et moins complexes) à résoudre. Cela représenterait un schéma électrique équivalent plus léger.

2.2 *Model order reduction* (*MOR*)

L'une des étapes d'un algorithme *MOR* est de connecter, à la place des composants externes, des sources idéales de tensions aux portes du circuit, puis un modèle mathématique l'assemble en utilisant l'approche *MNA* (*Modified Nodal Analysis* [4]) : une organisation logique sous forme matricielle des équations de *Kirchhoff* sur circuit, les KCL (*Kirchhoff's Current Law*) aux nœuds et les *KVL* (*Kirchhoff's Voltage Law*) aux branches, qui utilisent les tensions aux nœuds, les courants des inductances et des générateurs de tension comme variable d'état.
La simulation de circuit électrique est basée sur l'inversion matricielle dans le domaine fréquentiel, ou basée sur la résolution d'un système d'équations différentielles dans le domaine temporel. Au vu de la complexité croissante des schémas, ces approches, bien qu'optimisées, répondent de moins en moins aux contraintes industrielles de simplicité de mise en œuvre et de rapidité. Le circuit nécessite donc

du *MOR* efficace, précis, préservant au minimum sa passivité et ses caractéristiques électriques, tout en réduisant le nombre de ses variables d'états.

Cet algorithme de *MOR* [18], bien que préservant les caractéristiques électriques du circuit (en garantissant à l'origine du domaine de *Laplace*), est surtout applicable aux circuits dont les composants obligent à une prise en compte des forces électromagnétiques. L'effet *speed-up* a divisé de 2 à 2650 fois les temps de réponse selon les exemples.

Pour éviter de surcharger le simulateur dans le cas des *RLC* (très grands circuits irréguliers, ou constitués de grandes quantités interconnectées d'ensembles de résistance, bobine et condensateur), l'algorithme PartMOR [13, 17] partitionne ces hypergraphes *RLC* en sous-circuits de taille équivalente tout en minimisant les liens à couper pour les séparer. Cette approche ***netlist-in-netlist-out*** produit une *netlist* réduite et stable (étant un standard de données, cf. la partie 3), étant **l'un de nos verrous techniques résolus** par une modélisation alternative des données par fusion, mais ne s'applique qu'à un seul type complexe de circuit. [9, 14, 20] décomposent le circuit en sous-circuits remplacés par des *low-order macromodel* équivalents, puis les recombinent en un seul *reduced-order model*. Cette méthode a l'avantage d'utiliser la structure hiérarchique naturelle des circuits. Toutefois, un sous-circuit trop grand donnera un *low-order macromodel* trop imprécis pour l'équivaloir. À l'inverse, le macro-modèle d'un sous-circuit trop petit aura la même complexité que l'original, produisant une transformation plutôt qu'une réduction. La limite récurrente de chaque *MOR* est la réalisabilité du modèle réduit à un modèle mathématique ou une équation d'états (au lieu d'une *netlist*) et ceci pour un seul type de circuit. **Tout le processus d'analyse et de simulation est à modifier pour supporter ces nouvelles étapes** [8, 19] **à résoudre**, ralentissant les simulateurs plus qu'ils n'ont été accélérés en réduisant le modèle.

Nos schémas sont composés de circuits uniques, mais aussi de sous-circuits récurrents, ce qui nous oriente sur les règles de fusion s'y s'appliquant. [21] démontre l'efficacité du *MOR* sur les trois approches d'analyses de circuits ayant de grandes quantités de résistances connectées réduites en une seule équivalente. Un tel processus exige de : parcourir le graphe à la recherche de résistances connectées, calculer leurs équivalences et les remplacer. Cette approche s'appuie sur la théorie des graphes, l'algèbre linéaire numérique et des algorithmes de réordonnancement matriciel, afin de réduire fortement les circuits composés d'autant de résistances par un graphe équivalent de leurs nœuds externes et réduit de ceux internes [3, 20]. La simulation serait plus rapide et sans approximation. Des limites [11, 12] existent puisqu'un simulateur ne peut plus prévoir des effets sur un composant si son origine a été réduite.

Les sections sur le format de données et de leur fusion sont inclues dans la partie suivante de nos méthodologies d'accélération, puisqu'en les intégrant à *Emulatio*, nous avons redéfini / réinterprété certaines approches.

3 Accélérations pour l'émulation

Dans cette partie, nous présentons les différents modules d'*Emulatio* (la communication interopérable avec les autres systèmes de co-simulation, le simulateur intégré, la gestion des scénarios, *logs*, format de données du schéma, etc.) qui rencontrent différentes sous-problématiques d'accélération, dans lesquelles nous avons intégré **des processus d'intelligence augmentée** (*middlewares*, *shared memory*, *multi-agent system*, *expert system*) pour en améliorer les performances. Les dernières sections présentent notre **processus de fusion de données** basé sur du *complex event processing* qui accélère les temps de réponse du simulateur.

3.1 *Middlewares* et *Shared Memory*

Aucun expert n'acceptera de remplacer une architecture électrique par son jumeau numérique (malgré des limites connues telle qu'une obsolescence technique ou programmée) tant qu'il n'est pas certifié fiable, embarquable et interopérable. Pour cette communication multidirectionnelle de C&C avec les réels systèmes embarqués qui opèrent à la "vitesse des électrons", nous avons besoin de concevoir un **émulateur temps-réel** : calculant sans incohérence / défaillance les signaux analogiques et logiques, et les communiquant aux autres systèmes embarqués sans les faire attendre. Notre sous-**problématique** a été **l'accélération de ces communications**.

Jusqu'en 2016, nous avons menés des travaux R&D [22, 23, 24] sur la parallélisation interne (comme l'asynchronisation des modules par du *multithreading*) et externe par nos *Middlewares* nous amenant à créer notre librairie de mémoire partagée pour un échange d'information "traduit" hautement véloce entre *Kernel*, logiciels et systèmes embarqués. La Figure 2 indique chacun de nos *Middlewares* intègrant l'*API* de chaque logiciel de co-simulation. Ces intégrations nous a permis de déployer *Simulatio* depuis 2017 sur les bancs d'essais des *TrainLabs* accélérant la validation (jusqu'au jour où l'accélération et la certification d'*Emulatio* nous permettra de l'embarquer directement sur matériel).

Si aujourd'hui notre vitesse de communication externe reste surtout dépendante de la vitesse du simulateur, c'est depuis notre sous-**problématique d'accélération des communications internes des C&C** ci-dessous.

3.2 Système Multi-Agents C&C du Pupitre

Lorsqu'un utilisateur commande une action sur le Pupitre, si l'image de ce composant ne s'affiche pas instantanément (<300ms) comme à l'état (dés)activé alors le comportement de l'utilisateur est de relancer l'action. Cela lance une 2nd commande (dont le mode *on/off* annule la commande désirée, obligeant à en relancer une 3ème). Le module Scénario ou les *Middlewares* manipulant le Pupitre reproduisent aussi ce comportement une fois un temps d'attente dépassé (et parfois les relançant en grand nombre). De nombreux cas à simuler répètent les mêmes calculs (cf. clignotement de LED) et ne peuvent être évités automatique-

ment sans risquer des incohérences de simulation (telles que des commandes s'intercalant pendant le clignotement).

De la Figure 3 à 4, le Pupitre a été recodé en **un Système Multi-Agents** [26, 27] **de C&C *multithreads*** afin de s'exécuter en priorité sur les calculs matriciels du Simulateur (qui consomment la totalité des ressources d'un cœur du processeur). Chaque image (état d'activation d'un C&C) est géré par un agent réactif asynchrone qui communique ("1er envoyé 1er exécuté" comme en embarqué) par la technologie "*Signal & Slot*" et "*Process Events*" (*thread safe*) de la librairie *Qt*. Pour la gestion des messages de **commandes** (actions entrantes) et de **contrôles** (nouveaux états sortants), un *thread* centralise une machine d'état routant les messages (stockés dans une pile) à chaque agent concerné, qui l'interprète, agit ou non (commande une nouvelle valeur de ses paramètres au Simulateur) puis informe les modules du résultat (mettant à jour le Pupitre). Nous avons eu de 1 200 à plus de 16 000 agents C&C selon les projets (soit 300 à 700 Mo en RAM, cf. la partie 4).

3.3 Simulateur : l'agent *Gnucap*

SPICE (logiciel libre "*Simulation Program with Integrated Circuit Emphasis*") est le standard de simulation de circuits électroniques analogiques, permettant la simulation au niveau du composant (résistances, condensateurs, transistors, etc.) en utilisant différents types d'analyses : point de polarisation (courant continu), analyse fréquentielle pour petits signaux et bruit (courant alternatif linéaire), et transitoire. De nombreux logiciels sont des IHM intégrant *SPICE*.

Après une veille technologique des simulateurs et quelques preuves de concept, nous utilisons *Gnucap* (*"GNU Circuit Analysis Package"*) [5] comme simulateur (ou d'autres) pour sa prise en charge du format *Netlist* (standard de données d'un graphe électrique) et de simulation *SPICE*. Il peut effectuer une analyse *DC* (courant continu) et transitoire (*transient*) non linéaire (pilotée par les événements), une analyse de *Fourier* et une analyse *AC* (courant alternatif) linéarisée jusqu'au point de convergence. Il est pilotable en ligne de commande, interactif sur la sortie standard, exécutable en mode *batch* ou en tant que serveur.

Nos tests ont démontré la performance de *Gnucap* en termes d'imitation de la réalité électrique des schémas et pour sa capacité à répondre un résultat horodaté déterministe (des C&C lorsque les tensions de chaque nœud du graphe ont convergé) nous permettant la colorisation. Certains simulateurs analogiques fournissent ce résultat par un algorithme *anytime* (fournissant une solution à chaque unité de temps configurée), et comme les simulateurs logiques, ils peuvent "faussement imiter" (tels que l'activation binaire ou par dépassement de seuils) la réalité électrique dans nos fameux cas analogiques indéterminables logiquement (cf. les travaux antérieurs, puisque simulant les interactions et les tensions sans prise en compte des phénomènes physiques).

Un agent asynchrone charge dynamiquement *Gnucap* (ou un autre simulateur de notre choix) une fois les modules initialisés du *Kernel*. À la difficulté d'améliorer les performances de *Gnucap* (telles que la parallélisation de la décomposition *LU* sur GPU, plutôt que CPU, qui s'est révélée moins performante sur nos schémas composés de grands circuits), nous avons ainsi créé *Simulatrix* afin de le remplacer. Notre tentative s'est soldée par un simulateur logique que nous pouvons utiliser pour démonstration. Au final, nous avons réussi à accélérer *Gnucap*.

3.4 Vectorisation dans *Gnucap*

Afin de réaliser les estimations des valeurs de signaux électriques, *Gnucap* utilise principalement deux méthodes calculatoires : la décomposition *LU* (*lower upper*) [1] à l'aide de l'algorithme de *Crout* [6] pour factoriser la matrice des tensions aux nœuds, et une méthode de résolution des équations différentielles (cf. la section 2.1). Après analyse des temps d'exécution des différentes fonctions [7], il s'avéra que la majeure partie du temps de calcul est dépensée sur le calcul des séries de l'Algorithme 1 de *Crout* (ligne 3 et 9).

Algorithm 1 Décomposition *LU* d'une matrice A avec l'algorithme de *Crout*

1: **for** k = 1 : n **do**
2: **for** i = k : n **do**
3: $l_{ik} = a_{ik} - \sum_{p=1}^{k-1} l_{ip} u_{pk}$
4: **end for**
5: **if** k=n **then**
6: break
7: **end if**
8: **for** j = k+1 : n **do**
9: $u_{kj} = \frac{a_{kj} - \sum_{p=1}^{k-1} l_{kp} u_{pj}}{l_{kk}}$
10: **end for**
11: **end for**

De ce constat, **nous avons recodé ce calcul des séries** en utilisant des fonctions intrinsèques vectorisées (de la librairie *SSE4*) pour contourner les optimisations insuffisantes des compilateurs (après avoir testé différentes combinaisons de ses *flags* pour la plus performante). Les unités de calcul en virgule flottante des processeurs actuels peuvent multiplier et additionner 4 nombres en parallèle. L'utilisation de ces instructions ont permis ainsi de **réduire significativement ces temps de calcul de 30 %** en simulation.

Les ordinateurs servant de co-simulation en *TrainLab* ne sont pas connectés à internet (pour réduire le risque de fuite de données confidentielles). Une co-simulation se produit sur un seul PC (Windows à cause des logiciels propriétaires et des prix de leurs licences) dont le Pupitre est manipulé par des équipes de schématistes à tour de rôle en 3 fois 8 h jusqu'à 72 h par semaine, suivis par les corrections du schéma (ce qui oblige à maîtriser toutes fuites de mémoire, amplifiées par l'accélération). **Chaque pourcentage d'accélération augmente le nombre de scénarios validés en 72 h**, donc le respect des *deadlines* du projet.

Pour accélérer, nous avons optimisé la configuration matérielle. L'*overclocking* du CPU est apprécié, mais l'obsolescence matérielle combinée à son utilisation intensive en co-simulation peut achever le CPU avant la fin du projet. Sachant que la décomposition *LU* s'effectue sur un seul cœur (et que les autres cœurs sont utilisés par le SMA *multithread*), en termes de plus grande rapidité de calcul à moindre coût, nous avons expérimenté et préférons cet ordre de priorité : un CPU *overclocké* avec la plus haute fréquence, puis quelques cœurs ; deux RAM identiques pour le *Dual Channel* ayant surtout le plus petit *CL* (*cas latency*), puis la plus haute fréquence, version DDR et taille en Go ; un GPU dédié, pour ne pas limiter le CPU, et suffisant pour l'*Unity 3D* en co-simulation.

3.5 Le module *Netlist*

Pour fusionner des données, il nous faut partager notre traitement des données du format *Netlist* (un standard pour représenter les circuits électriques, cf. Figure 6), étant une description textuelle en langage prédéfini d'une liste d'éléments où chacun décrit : un nom d'instance, une référence au modèle, une liste des interconnexions de l'élément vers les nœuds du circuit, et une référence aux paramètres de l'élément. Des sous-circuits peuvent être ajoutés à la liste pour traiter leur hiérarchie. En théorie des graphes appliquée à un schéma électrique (cf. Figure 7), arêtes et sommets sont les composants interconnectés par des nœuds d'un graphe non orienté (puisque le courant se propage dans toutes les directions sur les dipôles non orientés).

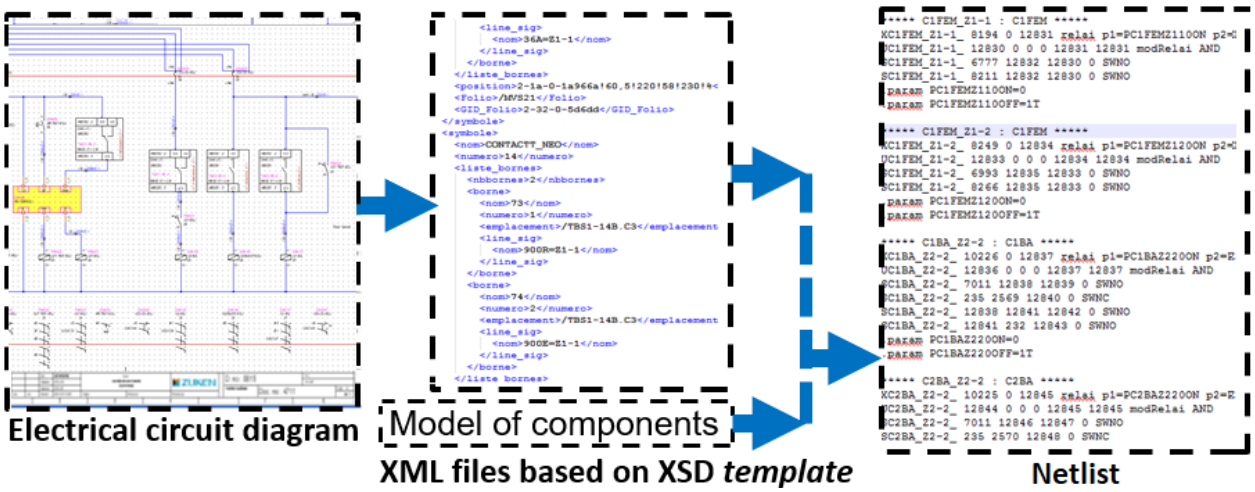


FIGURE 6 – Exemple d'une partie de *netlist* du schéma électrique issu du pipeline schématisé dans la Figure 2.

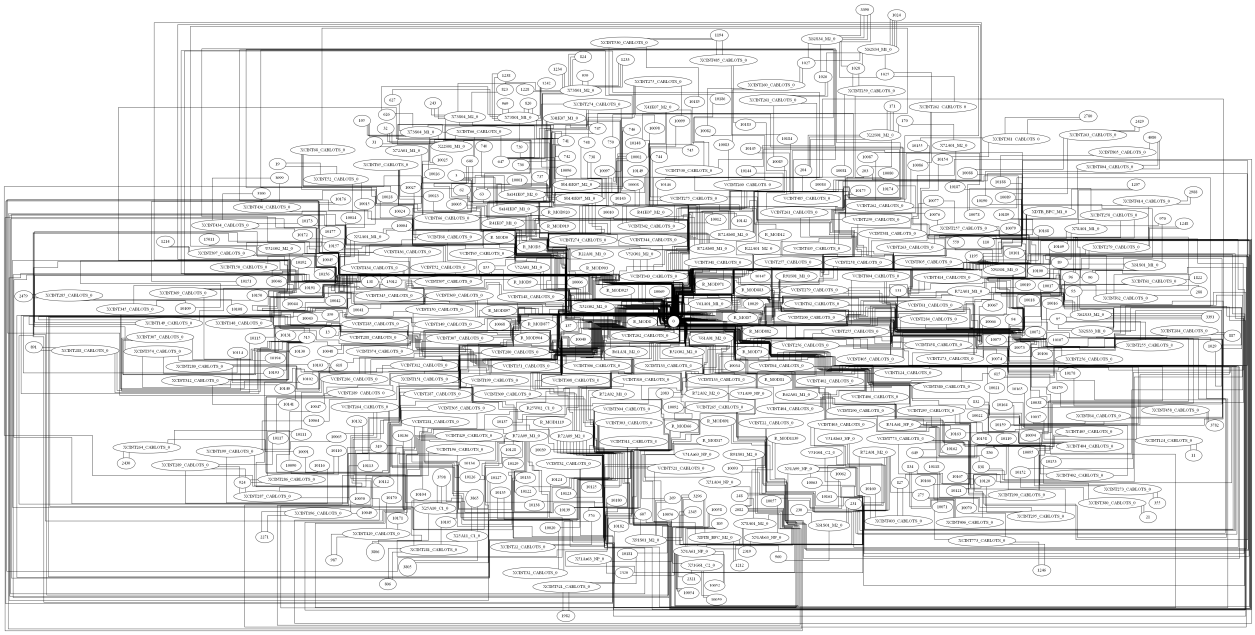


FIGURE 7 – Graphe de la *netlist* de la Figure 6 après fusion (puisque la totalité sans fusion n'est pas supportée sur le PC avec GraphViz). Les nœuds sont numérotés dont 0 au centre représente la masse, et les types de composants sont identifiables par leur 1ère lettre de *Netlist* (***R**esistor, **S**ubcircuit, **L**ogic device, **V**oltage source*, etc.).

Afin de modéliser les spécifications fonctionnelles uniques de C&C des projets, nous avons créé une grammaire syntaxique pour interpréter numériquement le contenu de *netlists* dans un *AST* (*abstract syntaxic tree*). Pour automatiser son interprétation en mémoire et son écriture en sortie après modification, nous avons utilisé les modules *Qi* (*parser* - analyseur syntaxique) et *Karma* (*generator*) de la librairie *Boost*. Différentes librairies existent pour concevoir sa propre grammaire syntaxique, cela reste très énergivore que ce soit en auto-génération (réclamant un modèle en entrée et une optimisation du code généré en sortie) ou manuellement (réclamant des maintenances du code). Comme nous n'en trouvons pas en version *open source* pour le format *Netlist* de *Gnucap* (n'étant pas celui de *Spice*), le *Listing* 1 présente notre grammaire syntaxique, pour votre usage, tout en incluant deux de nos modèles.
Au 1er lancement du projet, ***Emulatio* exécute l'approche *netlist-in-netlist-out*** : son système expert applique des règles de fusion (cf. les sections suivantes) sur le graphe *parsé* de la *netlist* et en génère une nouvelle sauvegardée pour lecture aux lancements suivants.

Listing 1 – Grammaire syntaxique *C++* du format *Netlist*

```
using boost::spirit::qi;
using boost::spirit::karma;
using boost::spirit::ascii;
using boost::spirit::utree;
namespace netlist_grammar {
template <typename Iterator, typename Skipper>
    struct netlist_grammar_definition :
        qi::grammar<Iterator, utree(), Skipper> {
            netlist_grammar_definition() :
                netlist_grammar_definition::base_type(start) {
using qi::int_;
using ascii::char_; // any or set of char, ex: char_("0-9a-fA-F")
using ascii::string;
using qi::eol; // the end of line (CR/LF and combinations thereof)
using ascii::alnum; // alpha-numeric
using ascii::digit; // numeric digits
using ascii::blank; // spaces or tabs
using qi::lexeme; // inhibit the skipper, ex: matches "00" and not "0 0"
using qi::repeat; // repeat(n,m)[attr] will repeat attr from n with m more

end_line = eol; // end of line
from1toN_ints = lexeme[int_ >> *int_]; // to parse integer as integer
// Values requiring a string parser such as "-0.1", "1e-006", "1E12"
string_value = lexeme[-char_('-') >> +digit >> -(char_('.') >> +digit)
    >> -(char_("Ee") >> -char_("-") >> +digit)];
/* Possible names of component ID */
_ID = lexeme[(alnum | char_("_-")) >> *(alnum | char_("_-"))];
node = from1toN_ints; /* IN / OUT nodes under voltage */
control_node = from1toN_ints; // node receiving / sending a voltage
/* "R_infini_ID 1 0 Pi" or "R_epsilon_ID 1 0 Pe" (the mass is the 0 node) */
p_value = (string_value >> -char_('K')) | _ID; // "Pi=1E12", "Pe=1e-006"
R_resistor = node >> node >> p_value;
/* "SWNO" (opened) / SWNC (closed by default without voltage)
    are examples of designed switchs, activable by a voltage from a relay */
model_name = string("SWN") >> char_("OC"); // "S_switch_ID 1 2 10 0 SWNO"
S_voltage_controlled_switch = node >> node
    >> control_node >> control_node >> model_name;
/* The logic devices (N)AND, (N)OR, (N)XOR has its ouput node activated
    by input nodes, ex: "U_logic_device_ID 1 0 0 0 2 3 AND" */
gatetype = -char_('N') >> (string("AND") | (-char_('X') >> string("OR")));
U_logic_device = node >> node >> node >> node
    >> control_node >> +control_node >> gatetype;
/* "V_battery_ID 1 0 Param_voltage" with ".param Param_voltage = 72" Volts */
V_independant_voltage_source = node >> node >> p_value;
/* "X_subcircuit_ID 1 0 relay p1=P_ON p2=P_OFF" for a relay model */
subname = string("diode_nominal") | string("relay");
x_parameters = char_('p') >> -digit; // "p1", "p2"
X_subcircuit_call = +node >> subname >> *(x_parameters >> char_('=') >> _ID);
/* The parsing works only on forward, without backward */
component_end = ( // The first letter of a line declares the component
    (char_('R') >> _ID >> R_resistor) |
    (char_('S') >> _ID >> S_voltage_controlled_switch) |
    (char_('U') >> _ID >> U_logic_device) |
    (char_('V') >> _ID >> V_independant_voltage_source) |
    (char_('X') >> _ID >> X_subcircuit_call)
);
/* All the possible lines of a netlist file */
// comments of a "***** NAME CIRCUIT *****"
stars_list = lexeme[repeat(4,5)[char_('*')]];
circuit_line = stars_list >> +_ID >> stars_list >> +end_line;
// ".param P_switch_ID = o" to declare an initial state of a switch
// Possible values of param such as 0, 1, o (opened), f (closed), etc.
id_param_ = _ID >> char_('=') >> (lexeme[+digit] | char_("of") | string_value);
_param_line = string(".param") >> id_param_ >> +end_line;
component_line = component_end >> +end_line;
start = +(circuit_line >> *component_line >> *_param_line);
/* "start" includes all lines of a netlist file */
BOOST_SPIRIT_DEBUG_NODE(start);
}
/* Design of the rule starting the netlist parsing process */
qi::rule<Iterator, utree(), Skipper> start;
/* Design of a rule iterating the data in an integer without skipper */
typedef qi::rule<Iterator, int()> rule_int;
/* Design of a rule iterating the data in utrees splitted by a skipper */
typedef qi::rule<Iterator, utree(), Skipper> rule_utree_skipper;
/* Design of a rule iterating the data in a string utree */
typedef qi::rule<Iterator, std::string()> rule_string;
rule_int from1toN_ints;
rule_string _ID, string_value, x_parameters;
rule_string subname, model_name, gatetype, end_line, stars_list;
rule_utree_skipper
    circuit_line, id_param_, node, control_node, p_value,
    R_resistor, S_voltage_controlled_switch,
    U_logic_device, V_independant_voltage_source,
    X_subcircuit_call, component_end, component_line, _param_line;
}; }
/* Design of an iterator type to iterate the netlist data */
typedef std::string::const_iterator iterator_type;
/* Design of a skipper type (to parse "\n", the ascii::blank_type is used
    as skipper instead of ascii::space_type */
typedef ascii::blank_type& skipper;
/* The netlist grammar definition according to a chosen iterator and skipper.
    Modify the skipper / iterator may totally change the resulting utree */
typedef netlist_grammar::netlist_grammar_definition<iterator_type, skipper>
    netlist_grammar_definition; // the definition that will be loaded
```

Concernant les données de netlist et de tout le pipeline (cf. Figure 6), il y a une problématique liée au contexte industriel : la co-simulation est un environnement "**coopétitif**" (coopératif et compétitif, collaborant et concurrent) de schématistes de différentes sociétés participant aux mêmes projets collaboratifs. Lorsque des corrections sont à produire, certains utilisateurs modifient directement les données brutes sans utiliser les logiciels du pipeline (cf. Figure 2). Un seul système, logiciel ou modèle "faussé" par de telles modifications provoquera des incohérences en co-simulation ou même du chômage technique en cas de *bug* non résolu. **De l'intelligence augmentée tirée du *Q-Learning*** a été codée : les prochains utilisateurs sont notifiés où sont situés les modifications inattendues (pour y relancer ou non la suite du pipeline). Sa politique optimale est : (1) d'apprendre par itération les fautes, et selon les récompenses et les facteurs, (2) de les éviter en relançant automatiquement les fusions et/ou en chiffrant les données dont les modifications engendrent des fautes. Cela a fortement réduit le nombre de maintenance logicielle à distance. Les accélérations ont aussi fortement réduit le temps d'initialisation de co-simulation (de +30 min à -10 min).
Comme évoqué, nous nous sommes attaqués à réduire les temps de réponse en parallélisant les processus qui sont naturellement décomposés dans cette simulation afin d'atteindre l'émulation. Cette pratique a démontré un facteur 10 de diminution des temps de réponse, mais nous sommes encore loin de la "vitesse d'électron en embarqué".

3.6 Fusion de données et *CEP*

L'approche la plus élémentaire pour obtenir des informations de plus haut niveau sur les entités présentes dans un environnement observé est la fusion de données du standard *JDL Data Fusion* [10, 15, 25], issu de l'*U.S. Joint Directors of Laboratories Data Fusion Group*, qui définissent la fusion de données comme étant ("traduite") : "*Un processus traitant l'association, la corrélation, et la combinaison des données et l'information des sources simples et multiples, pour réussir à raffiner la position, les évaluations identitaires, et les estimations complètes et opportunes des situations, des menaces et leur signification. Le*

processus est caractérisé par des mises au point continues de ses évaluations et estimations, et l'évaluation du besoin de sources additionnelles, ou de la modification du processus lui-même, pour réaliser des résultats améliorés."

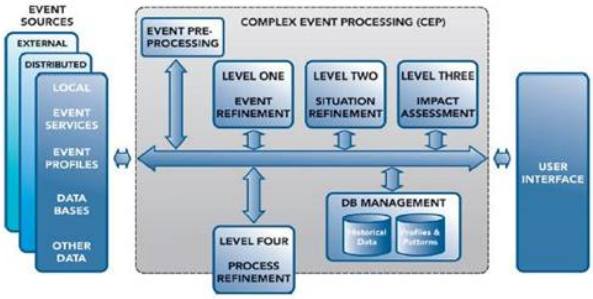


FIGURE 8 – *Complex Event Processing (CEP) by JDL Data Fusion* (source : tibco.com)

Outre le prétraitement de chaque information bas niveau, ces dernières doivent être transmises aux échelons supérieurs pour agrégation avec d'autres informations en vue d'une analyse globale et d'une éventuelle décision d'action. Le modèle *JDL* affiché dans la Figure 8 sous la forme d'un *CEP* en ferait un modèle fonctionnel de fusion à interpréter selon le besoin, défini sur plusieurs niveaux de traitement de l'information. Ainsi, **pour chaque définition** ("traduite") **suivante d'un niveau du modèle, nous rajoutons l'interprétation que nous avons produit pour répondre à notre problématique de fusion de schéma électrique :**

- Niveau 0 : *Data assessment* - L'évaluation des données : l'estimation et la prédiction des états observables des signaux / objets en se basant sur une association de données au niveau du signal (étant une donnée brute émise du capteur physique ou logiciel). **Les données de *netlist* sont interprétées avec notre grammaire syntaxique.**
- Niveau 1 : *Object assessment* - L'évaluation des entités : l'estimation et la prédiction des états des entités en se basant sur l'association de données, l'estimation des états continus et discrets. **Pour chaque entité** (composant), **nous agrégeons toutes ses informations** (*casted data* de la *netlist*) **afin d'en estimer sa liste d'états possibles** (initial, activé, désactivé, etc.), **d'en prédire les conditions de changements d'états** (tensions, etc.), **ce qui le commande** (nœuds, paramètres), **ce qu'il contrôle** (idem), etc.
- Niveau 2 : *Situation assessment* - L'évaluation de la situation : l'estimation et la prédiction des relations entre les entités, y compris la structure des forces et les rapports de force, les communications, etc. **Pour un motif récurrent de relations entre entités** (de composants interconnectés), **nous générons des tentatives de règles qui pourraient fusionner ses relations et ses entités. Sont uniquement autorisées celles qui sont prédites comme permettant une moindre complexité des entités et/ou des relations internes au motif, et surtout qui ne modifie pas les relations externes du motif** (étant ce qui commande les composants du motif, les états qu'ils contrôlent et les composants externes qui sont commandés par le motif).
- Niveau 3 : *Impact assessment* - L'évaluation de l'impact : l'estimation et la prédiction des effets sur les situations des actions planifiées ou estimées par les participants, y compris les interactions entre les plans d'action des multiples acteurs. **Chaque règle est expérimentée par itération** (des circuits simples aux projets réels) **pour en estimer l'impact sur des situations** (des scénarios simples générés puis ceux enregistrés en co-simulation). **Sont uniquement autorisées celles qui n'ont généré aucune incohérence et qui ont accéléré la simulation.**
- Niveau 4 : *Process refinement* - L'amélioration des processus (un élément de la gestion des ressources) : l'acquisition et le traitement de données à mettre à jour pour soutenir les objectifs de détection. **Les motifs et les règles tentées sont sauvegardés pour ne plus être retenter. Pour chaque règle qui n'a pas été autorisée au niveau 3, des tentatives de correction sont faites** (par une modification de ses conditions d'activation lorsqu'elle a généré des incohérences, par un réordonnancement des règles qui n'ont pas accéléré la simulation, etc.). **Une détection est tentée pour évaluer si les motifs et les règles qui n'ont jamais été autorisés au niveau 2 et 3 ont statistiquement des informations en commun** (sous-relations ou sous-entités) **qui sont à bannir des prochaines tentatives, ce qui réduit le temps d'attente pendant les calculs.**

Pendant un scénario, chaque modification de paramètres (*On/Off*) entraînent des modifications dans le schéma électrique à recalculer. Ce sont des évènements qui engendrent des faits et peut-être d'autres événements qui, fusionnés, seront alors des événements complexes. Dans l'objectif d'éviter des incohérences de tensions, nous devons procéder à du traitement d'évènements complexes (*Complex Event Processing*, *CEP*) [16], telle la Figure 8 sur le modèle *JDL Data Fusion*. C'est un cas de Système Expert, une architecture logicielle pour l'aide à la décision, permettant de reproduire les mécanismes cognitifs d'experts. Pour la simulation d'un circuit électrique en *CEP*, il faut réunir :

- une **base de faits** (dans une *netlist*),
- une **base de règles** (de fusion),
- et un **moteur d'inférence** (un mécanisme de logique formelle tel que la déduction par chaînage avant, l'abduction par chaînage arrière, et mixte si les deux).

L'induction est rarement intégrée, étant la découverte automatisée de règles (cf. par "raisonnement par récurrence" ou "apprentissage par renforcement"), **que notre approche tente de produire**.
Dans cette section, nous avons évoqué l'induction de règles de fusion *MOR* adaptables à tout sous-circuit exécutable par un système expert de *CEP* sur tout le schéma. Afin d'éviter les incohérences lors des événements de scénarios, un système expert pourrait décomposer de l'induction : générer des composants interconnectés proches, puis les simuler par différents scénarios. Les motifs équivalents déterminés par analyse de circuit et plus rapide à simuler nous permettent de créer de telle règle de fusion.

3.7 Système expert à base de règles de fusion

Notre document interne sur notre processus de fusion de données (utilisant tous les fonctionnalités des modules décrits dans les sections précédentes et la section suivante) **fait plus d'une centaine de pages, donc nous le résumons ici avec des exemples dans la partie 4 :**

1. Recherche des sous-motifs fréquents de composants interconnectés dans chaque nouveau projet (ayant des spécifications fonctionnelles uniques) : des heuristiques *banlistent* les ensembles comportant des composants C&C du Pupitre pour ne pas les fusionner.

2. Pour chaque ensemble fréquent, leurs nœuds sont classifiés comme internes (ceux interconnectés) ou externes (ceux potentiellement alimentés ou à la masse), et les paramètres de C&C changeant leurs états sont listés.

3. Sauvegarde des résultats dans une machine d'états de tous les cas possibles après simulation de : chaque nœud externe alimenté par chaque gamme de tensions (0 la masse et de 12 à 110 V), et chaque scénario possible (donc toute séquence de paramètres puisque certains déclenchent nos fameux cas analogiques indéterminables logiquement).

4. Génération de motifs équivalents par une approche *anytime* inspirée de QCSP [2] (*quantified constraint satisfaction problem*, pour des heuristiques élaguant la recherche *Expspace*) jusqu'à obtenir (ou non selon l'horizon maximal configuré) une (ou plusieurs) machine(s) d'états égales. En exemple de contraintes quantifiées : la sauvegarde évite de les retenter, une estimation de probabilité évite de fouiller vers les branches en sur-échec, etc. Ce processus de "*data farming*" est très énergivore, mais nous obtient de nombreux résultats.

5. Génération de chaque règle de fusion qui produirait la transformation du motif fréquent en un motif équivalent.

6. Expérimentations itératives semi-automatisées : (1) fouille du graphe pour appliquer une de ces règles un maximum de fois localement (donc le 1er passage est total, puis sur les parties transformées et voisines, puisque un motif peut réapparaître par les transformations de toute règle), (2) simulation de scénarios et comparaison des logs pour vérifier les cohérences, sinon la règle est *blocklistée*, et (3) la règle est *grantlistée* si accélération prouvée.

7. Durcissement des conditions d'activation des règles *blocklistées* avec expérimentations itératives (pour la *grantlistée* si réussite sinon elle est *banlistée*).

8. Expérimentation de l'intégration de chaque nouvelle règle dans notre système expert *CEP* parmi celles déjà intégrées. La structure en variable chainée de l'*AST* de *netlist* nous empêche de paralléliser l'application de ces règles puisque générant des conflits de mémoire pour chaque nœud modifié par deux règles s'y appliquant. En intégrant une gestion par *mutex* (*thread safe*), leurs applications en parallèle s'est révélées plus lente qu'en séquentiel.

9. Expérimentation semi-automatisée de l'ordonnancement des règles pour fusionner un maximum le graphe. Le calcul du meilleur ordonnancement nous a exigé des heuristiques pour éviter de tester les cas incongrus (comme les cycles, tels que deux règles s'exécutant à l'infini lorsque l'une transformant un motif dans celui de l'autre). De plus, le graphe le plus fusionné (exigeant le plus de temps de calcul) ne produit pas les plus faibles temps de réponse.

Pour valider les accélérations et les cohérences de simulation pendant toutes ces expérimentations, nous nous sommes appuyés sur les modules suivants.

3.8 Scénario, Système temps-réel et *Logs*

Le module Scénario, en amélioration continue, automatise les actions récurrentes des experts schématistes. Actuellement, il permet de cloner les commandes reçues et d'ajouter les tests (de valeurs de tensions sur signal et d'état d'activation des composants) que le schématiste effectue visuellement sur la CAO colorisée. En cas de *bug* ou d'incohérence rencontrée pendant une co-simulation, il rejoue les mêmes actions du *log* généré par un agent asynchrone.
Comme nous devons mesurer les effets de chaque développement expérimental, le module Système temps-réel nous permet de déployer des agents asynchrones qui horodatent les actions à la microseconde par "*Posix Time*" de la librairie *Boost*. Le comparateur de *logs* génère les statistiques (cf. la partie 4) et identifie toute différence de tensions, puisque les fusions transforment le graphe, donc sensiblement les tensions. Nous acceptons (ou non) ces différences par une classification comme étant :

- équivalent : tant que $< X\ \% < 30\ \%$ avec X la tension attendue, dans la limite des gammes possibles de 12, 24, 48, 72 et 110 V (sachant qu'une différence de 50 % indiquerait l'apparition d'un pont diviseur de tension),
- une incohérence refusée : comme un nœud sous tension plutôt que 0 V et inversement.

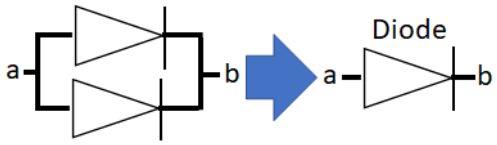
En remontant le *log* de fusion, la règle de fusion à l'origine de toute incohérence est *banlistée* ou corrigée (telle qu'augmenter ces conditions d'activation).

4 Résultats d'expérimentation

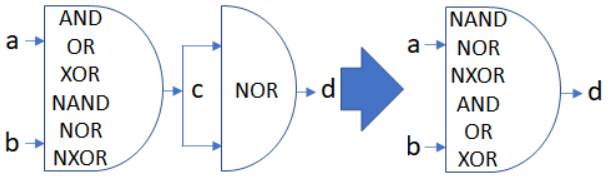
Notre modélisation alternative des données du schéma électrique s'effectue par un grand nombre de règles de fusion (cf. Figure 9) ayant des conditions d'activation qui leurs sont propres : nœuds, types et nombre de composants (parmi résistances, relais, bobines, portes logiques, contacts, paramètres C&C, etc.) interconnectés en série / parallèle / étoile / triangle, ... et/ou en nœud de C&C. Pour chaque règle exécutée, le système expert retentera les règles sur les parties transformées (et interconnectées) du graphe. De nombreux faits de bas niveaux sont agrégés en de moins nombreux faits de haut niveaux (cf. *CEP*).
Ces règles ordonnancées sont de deux types :

- **Réduction** : de N composants en 1 à $N-1$ équivalents (cf. les sous-figures 9a, 9b et 9c),
- **Transformation** : de N composants en N à $N+M$ équivalents (cf. la sous-figure 9d).

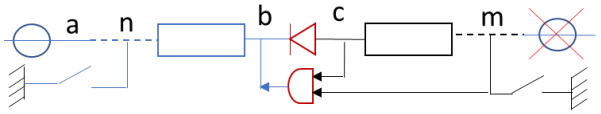
Notre ordonnancement le plus performant ressemble à une double boucle imbriquée se relançant jusqu'à un horizon maximal : lorsque les réductions ne s'activent plus, chaque transformation (qui relance statistiquement les réductions) est appliquée.



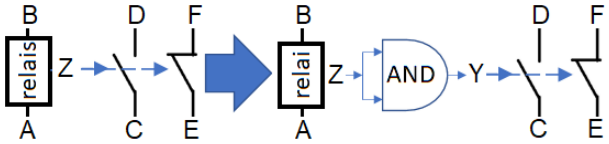
(a) Ces "diodes nominales" en parallèle dans le même sens sont agrégées en une seule (puisque seul la tension du nœud a alimente le nœud b)



(b) Toute porte logique (AND, OR, XOR, $NAND$, NOR ou $NXOR$) en série d'une NOT (étant une NOR à double entrée) peut être remplacée par une contraire.



(c) Séparés (en rouge) par une diode nominale (c alimentant b) et une porte logique (m et c alimente b ou non selon l'effet logique), les composants à gauche (alimenté en a) n'alimenteront jamais ceux à droite. Erreur schématique ou choix temporaire, la source de tension (rond barré en m) n'existe pas, donc tous les composants rouges et à leurs gauches sont agrégables en une seule résistance équivalente connectée à la masse.



(d) Les résultats montrent qu'en insérant une porte logique entre les modèles de relais et les contacts (CD, EF) activés (ou non) par le champ magnétique Z (calculé dans *Gnucap* par une équation différentielle en fonction des paramètres de la bobine du relais qui se charge analogiquement par la tension (stable ou sinusoïdale) entre A et B), alors le temps de charge de la bobine devient instantané.

FIGURE 9 – Différents exemples de fusions de sous-circuits en leurs équivalents. Ces règles sont les plus "simples" à interpréter pour les non experts. Statistiquement, ces règles de réduction s'exécutent lors d'erreurs de conception (sauf pour 9d). Toute ligne de *netlist* en moins allège la taille matricielle et accélère *Gnucap*.

4.1 Analyses des fusions sur *netlist*

Afin d'étudier l'effet d'une nouvelle règle, nous mesurons la complexité du graphe puisque les nœuds deviennent hyperconnectés par fusion. Dans la Figure 12, la fusion a augmenté le taux de 6 % (du nombre de connexions par nœuds), la densité de 221 %, et l'indice iota de 17 %. La densité est basée sur la formule $d = \frac{1000.|E|}{|V|.|V-1|}$ sachant V les sommets et E les arêtes d'un graphe non orienté $G = (V, E)$. Toutefois, elle fournit la même densité quand le schéma contient deux graphes de tailles différentes ayant le même taux. Pour y remédier, nous utilisons l'indice iota : un ratio entre la longueur du graphe et le nombre de nœuds, pondérés par leur importance (étant doublé lorsque >2 composants).

Nombre	Netlist originelle	Netlist fusionnée	Gain
Resistors	5162	**306**	**94%**
Components	1980	1488	**25%**
Subcircuits	1316	936	**29%**
Logic devices	152	152	0%
Vcswitchs	495	383	**23%**
Vsources	17	17	0%
Comments	1802	724	**60%**
Params	1199	1022	**15%**
Netlist AST cells	67184	25330	**62%**
Steps	0	43	
Reduction process time	0	11,3	
Nodes	3752	**1240**	**67%**
Alone nodes	409	**99**	76%
Links between elements	11000	3847	**65%**
Links to the mass	4580	813	**82%**
Average links/nodes	2,93	3,1	-6%
Density / 1000	1,56	**5,01**	**-221%**
Indice iota > 2	4,64	**5,41**	**-17%**

FIGURE 12 – Statistiques du contenu de la *netlist* du Projet 2 avant et après fusion, incluant les nombres de : résistances, composants (incluant relais, portes logiques, contacts, sources de tension, etc.), paramètres, cellules de l'*AST* (stockant la *netlist*), nœuds (incluant ceux seuls et connectés à un seul composant en schéma incomplet), connexions entre les nœuds (sans inclure les connexions à la masse). Le graphe a été parcouru 43 fois par toutes les règles en 11 secondes.

Statistics from pairs of subcomponent types being in parallel:				
Parallel of { RELAIS , RELAIS }	= 78	→ 61		
Parallel of { RELAIS , RESISTOR }	= 23	→ 21		
Parallel of { VSOURCE , RESISTOR }	= 1			
Statistics from pairs of subcomponent types being in serie:				
Serie of { DIODE_NOMINAL, C1TR }	= 70	→ 64	→ 43	→ 44
Serie of { LOGICDEVICE , LOGICDEVICE }	= 102	→ 95		
Serie of { RELAIS, LOGICDEVICE }	= 74	→ 48		→ 32
Serie of { RESISTOR, C1TR }	= 72		→ 71	
Serie of { RESISTOR, RELAIS }	= 0	→ 2		
Serie of { SWNOC , C1TR }	= 14	→ 0		
Serie of { SWNOC , DIODE_NOMINAL }	= 33	→ 31		
Serie of { SWNOC , LOGICDEVICE }	= 155	→ 141		
Serie of { SWNOC, RELAIS }	= 8			
Serie of { SWNOC , RESISTOR }	= 22	→ 5		
Serie of { VSOURCE, C1TR }	= 4	→ 0		→ 1
Serie of { VSOURCE, DIODE_NOMINAL }	= 4	→ 8		→ 0
Serie of { VSOURCE, LOGICDEVICE }	= 107			→ 91
Serie of { VSOURCE , RESISTOR }	= 4	→ 0		

FIGURE 13 – Statistiques du nombre de paires de types de composants connectés en parallèle et en série, contenues dans la *netlist* du Projet 2. Les trois colonnes de droite affichent l'effet de chaque nouvelle génération de règles *grantlistées* sur les pairs. Les C1TR / SWNOC sont des contacts activables par une commande utilisateur / relais.

L'observation de l'évolution de l'effet des règles, comme dans la Figure 13, nous a permis d'imaginer le graphe comme un arbre dont les racines sont les sources de tensions, dont les fusions élaguent les feuilles (la masse) jusqu'aux branches (sous-circuits), diminuant le besoin en entretien du paysagiste (simulateur).

4.2 Estimation des accélérations

Affiché dans la Figure 10, chaque calcul est ordonnancé par une commande (noire), un résultat (vert) transmis au SMA, modules et colorisation, et un "Total" (bleu) des temps de réponse combinant simulation et transmission. Ces "Total"s nous affiche un facteur 101 d'accélération pendant la 1ère partie d'un scénario s'initialisant en deux temps (chargement et initialisation) suivi de cinq commandes : fermant

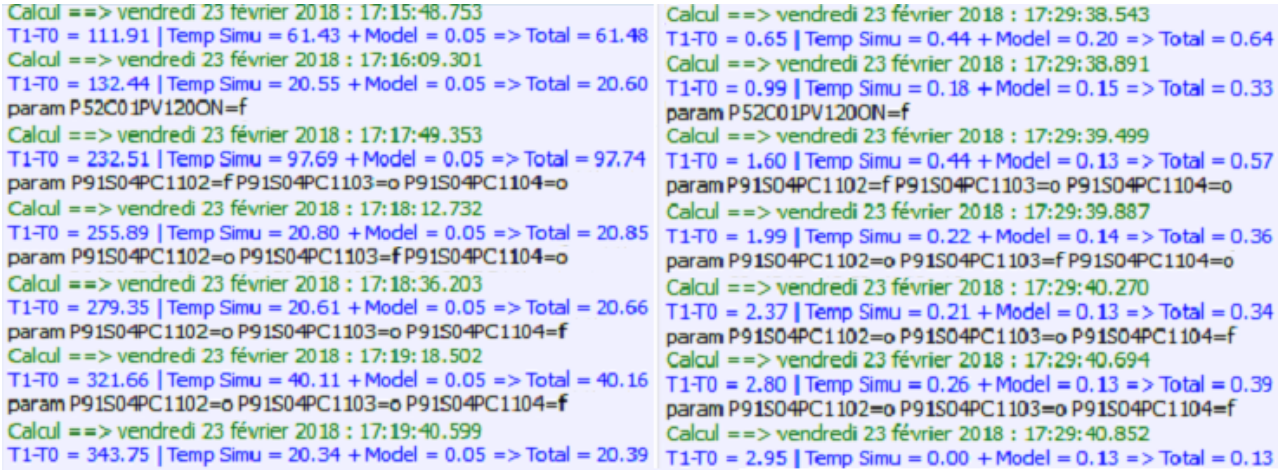


FIGURE 10 – Des temps de réponse d'*Emulatio* en seconde (sans fusion et avec fusion) du Projet 2 (cf. Figure 7) sur banc de test. L'accélération du simulateur ("*Temp Simu*") entraînait de trop grande quantité de communication entre les modules asynchrones, ralentissant le temps ("*Model*") de transmission des données. **Le SMA des C&C** a permis de l'accélérer au prix d'une consommation mémoire, afin qu'il ne s'ajoute plus à la somme "*Total*" du temps d'attente vécu par les utilisateurs.

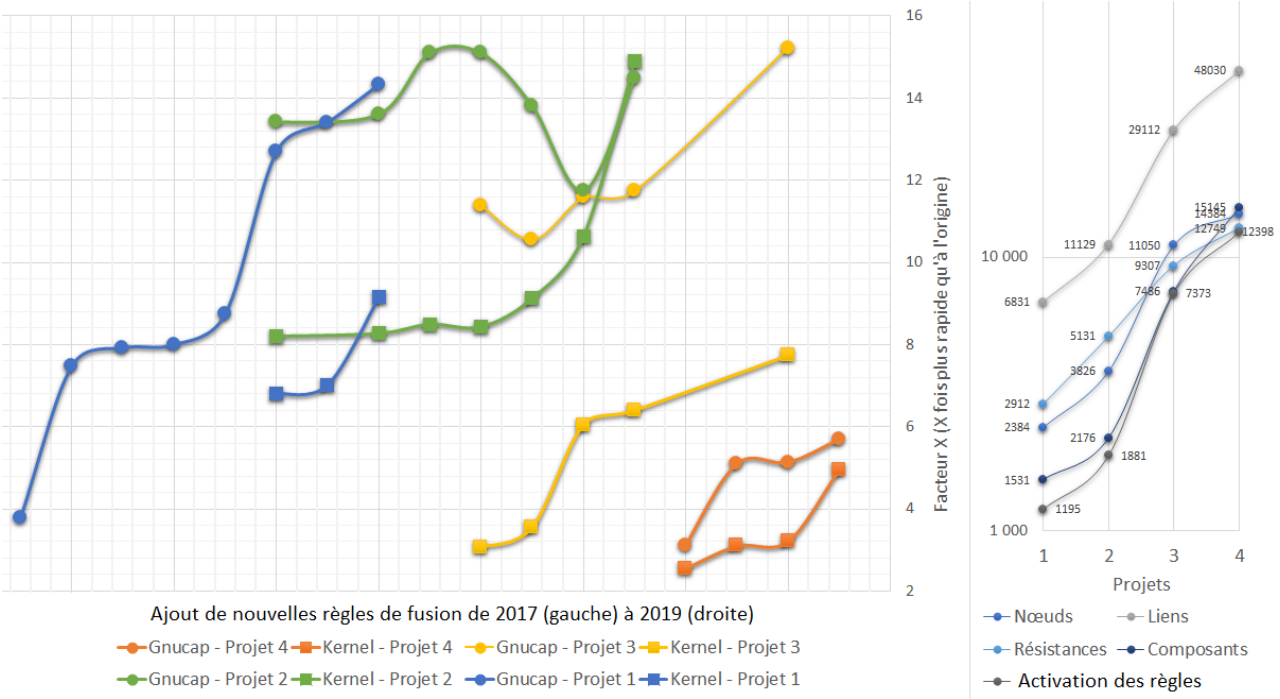


FIGURE 11 – La complexité des schémas de 4 projets de 2017 à 2019 à droite, et à gauche leur évolution d'accélération des temps de réponse (en ordonnée Y fois plus rapide) sur différents scénarios de plusieurs milliers de commandes (répétés 3 à 5 fois) sur les bancs de test des *TrainLabs*.

un contact pour alimenter un relais, forçant son activation, le libérant (s'activant si tension suffisante) et forçant deux fois sa désactivation (dont le 2nd résultat ne changera pas, réduisant son "Temp Simu" à 0 sec, étant une forme d'accélération supplémentaire).

Chaque projet a des spécifications fonctionnelles uniques, donc l'accélération obtenue n'est pas identique, comme affichée à gauche dans la Figure 11 avec les courbes de ronds pour *Gnucap* et de carrés du *Kernel*. De 2017 à 2019, chaque projet a multiplié la complexité du graphe, comme affichée à droite dans la Figure 11 en nombre de : nœuds, liens entre les nœuds (sans ceux à la masse), résistances (responsables des plus gros ralentissements), autres composants et activations des règles. Des nouvelles versions d'un schéma provoquent parfois quelques décélérations, surtout lorsque des règles ont été désactivées pour des incohérences découvertes. Nos travaux R&D en continu cherchent à améliorer nos algorithmes (cf. section 3) qui nous permettent de ré-accélérer par de nouvelles règles.

Les quatre projets de la Figure 11 ont été déterminants pour optimiser la transmission bidirectionnelle des données entre les modules d'*Emulatio*, puisque de telles accélérations ont provoqué une infobésité à transmettre (cf. Figure 2).

Remerciements, bilan et perspectives

Les auteurs et de nombreux collaborateurs (>40 devenus experts, parmi *Adrien*, *Alexandre*, *Anne-Sophie*, *Ludovic*, *Stéphane*, *Emmanuel*, *Florent*, etc.) de *Ccamy Systèmes* et sur *TrainLabs* se sont démenés (*versioning de code*, application des 12 facteurs, veilles et preuves de concept, spécifications, expérimentations, tests fonctionnels, management Kanban R&D, etc.) pour accélérer *Emulatio* (et la validation CAO), par fusion de données et intelligence augmentée. **Les équipes de *TrainLabs* ont réduit de 60 % les essais sur prototype qui représentent habituellement 15 % du coût de tel projet.** Nous espérons participer ainsi à la transformation numérique des processus des schématistes, à une amélioration de leur formation et à la compétitivité ferroviaire française.

L'accélération actuelle soulage les cœurs du processeur, ce qui nous permet d'envisager plusieurs instances du simulateur en parallèle, dont les répliques calculeront les commandes futures les plus probables estimées par *machine learning* des logs des scénarios passés. Pour la "vitesse d'électron" d'un émulateur temps-réel, le facteur 1000 (<10ms) est à atteindre, afin de pouvoir embarquer le jumeau numérique de toute architecture électrique à valider ou même à remplacer.

En dehors du domaine ferroviaire, *Emulatio* pourrait s'appliquer sur les systèmes contrôle-commande embarqués de nombreux domaines : nucléaire, modélisation des informations du bâtiment (*BIM*), réseau électrique intelligent (*smart grid*), objets connectés (*IoT*), véhicules, etc.

Références

[1] R. H. Bartels and G. H. Golub. The simplex method of linear programming using lu decomposition. *Communications of the ACM*, 12(5) :266–268, 1969.

[2] M. Benedetti, A. Lallouet, and J. Vautard. Qcsp made practical by virtue of restricted quantification. In *IJCAI*, pages 38–43, 2007.

[3] M. H. Chowdhury, C. S. Amin, Y. I. Ismail, C. V. Kashyap, and B. Krauter. Realizable reduction of rlc circuits using node elimination. In *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.*, volume 3, pages III–III. IEEE, 2003.

[4] L. O. Chua, C. A. Desoer, and E. S. Kuh. *Linear and nonlinear circuits*. 1987.

[5] A. Davis. Gnucap-the gnu circuit analysis package, 2015.

[6] A. T. Davis. Implicit Mixed-Mode Simulation of VLSI Circuits.

[7] A. T. Davis. An overview of algorithms in gnucap. In *Proceedings of the 15th Biennial University/Government/Industry Microelectronics Symposium (Cat. No. 03CH37488)*, pages 360–361. IEEE, 2003.

[8] A. Devgan and P. R. O'Brien. Realizable reduction for rc interconnect circuits. In *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No. 99CH37051)*, pages 204–207. IEEE, 1999.

[9] R. W. Freund. Sprim : structure-preserving reduced-order interconnect macromodeling. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 80–87. IEEE, 2004.

[10] D. L. Hall, M. E. Liggins, and J. Llinas. *Handbook of multisensor data fusion : Theory and practice*. CRC press, 2008.

[11] D. Harutyunyan, R. Ionutiu, E. J. W. ter Maten, J. Rommes, W. H. Schilders, and M. Striebel. Advanced topics in model order reduction. In *Coupled Multiscale Simulation and Optimization in Nanoelectronics*, pages 361–432. Springer, 2015.

[12] R. Ionutiu. Model order reduction for multi-terminals systems : with applications to circuit simulation. 2011.

[13] G. Karypis. Metis, a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 4.0. *http ://glaros. dtc. umn. edu/gkhome/metis/metis/download*, 1998.

[14] Y.-M. Lee, Y. Cao, T.-H. Chen, J. M. Wang, and C.-P. Chen. Hiprime : Hierarchical and passivity preserved interconnect macromodeling engine for rlkc power delivery. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6) :797–806, 2005.

[15] J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White. Revisiting the jdl data fusion model ii. Technical report, SPACE AND NAVAL WARFARE SYSTEMS COMMAND SAN DIEGO CA, 2004.

[16] D. Luckham. The power of events : An introduction to complex event processing in distributed enterprise systems. additions-wesley. *Reading*, 2001.

[17] P. Miettinen, M. Honkala, J. Roos, and M. Valtonen. Partmor : Partitioning-based realizable model-order reduction method for rlc circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(3) :374–387, 2011.

[18] A. Odabasioglu, M. Celik, and L. T. Pileggi. Prima : Passive reduced-order interconnect macromodeling algorithm. In *The Best of ICCAD*, pages 433–450. Springer, 2003.

[19] T. Palenius and J. Roos. Comparison of reduced-order interconnect macromodels for time-domain simulation. *IEEE transactions on microwave theory and techniques*, 52(9) :2240–2250, 2004.

[20] Z. Qin and C.-K. Cheng. Realizable parasitic reduction using generalized y-δ transformation. In *Proceedings of the 40th annual Design Automation Conference*, pages 220–225, 2003.

[21] J. Rommes and W. H. Schilders. Efficient methods for large resistor networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(1) :28–39, 2009.

[22] B. Six. *Génération automatique de modèles pour la supervision des systèmes dynamiques hybrides : application aux systèmes ferroviaires : Automated Model builder for supervision of Hybrid Dynamic Systems : Applied on a railway rolling stock system*. PhD thesis, Université Lille, 2018.

[23] B. Six, B. Ould-Bouamama, S. Lintz, and A.-L. Gehin. Railway direct braking system analysis using hybrid bond graph. In *2016 International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 034–039. IEEE, 2016.

[24] B. Six, B. Ould-Bouamamat, S. Lintz, and A.-L. Gehin. Automated model builder of hybrid bond graph : Application to a two level inverter. In *2017 IEEE 11th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*, pages 325–330. IEEE, 2017.

[25] A. N. Steinberg and C. L. Bowman. Revisions to the jdl data fusion model. In *Handbook of multisensor data fusion*, pages 65–88. CRC press, 2017.

[26] R. Stuart, N. Peter, et al. Artificial intelligence : a modern approach, 2003.

[27] M. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.