

Vérification de politiques d'apprentissage par renforcement dans des environnements non linéaires à l'aide de polytopes

RAPPORT DE STAGE DE RECHERCHE

24 juillet 2025



Table des matières

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 2 | Problématique et contexte scientifique | 4 |
| 2.1 | Contexte du travail | 4 |
| 2.2 | Présentation du problème | 4 |
| 2.2.1 | Présentation de <i>Cartpole</i> | 4 |
| 2.2.2 | L'apprentissage par renforcement | 5 |
| 2.3 | État de l'art et analyse critique | 7 |
| 2.3.1 | Vers des politiques <i>RL</i> interprétables | 7 |
| 2.3.2 | Méthodes de vérification formelle | 7 |
| 2.3.3 | Défis ouverts | 8 |
| 2.4 | Travaux réalisés précédemment | 8 |
| 2.5 | Contributions | 9 |
| 2.5.1 | Déroulement de la recherche | 9 |
| 3 | Travail réalisé | 10 |
| 3.1 | Préambule : Représentation des états et objectifs | 10 |
| 3.2 | Phase 1 – Reproduction du cas linéaire sur un seul chemin | 10 |
| 3.2.1 | Hypothèses et méthodes | 10 |
| 3.2.2 | Présentation d'une étape | 12 |
| 3.3 | Phase 2 – Passage au cas non linéaire, suivi d'un seul chemin | 12 |
| 3.3.1 | Présentation des changements théoriques | 12 |
| 3.3.2 | Encadrement octogonal (cas non-linéaire) | 12 |
| 3.3.3 | Antécédents approximés (cas non linéaire) | 17 |
| 3.3.4 | Affinement itératif des octogones | 18 |
| 3.3.5 | Affinement avec contrôle fin du ϵ : problème de la borne trop lâche | 19 |
| 3.4 | Phase 3 – Suivi de l'ensemble des chemins | 20 |
| 3.4.1 | Nouveaux outils théoriques | 20 |
| 3.4.2 | Clusterisation des polytopes | 21 |
| 3.4.3 | Suppression des polytopes déjà-vus (Récursion) | 23 |
| 3.4.4 | Expérimentations du clustering et résultats sur la récursion | 24 |
| 3.4.5 | Algorithme général de vérification <i>CartPoleRL-Safe</i> | 26 |
| 4 | Résultats | 27 |
| 4.1 | Certification d'un chemin en physique non-linéaire | 28 |
| 4.2 | Certification globale en physique non-linéaire | 29 |
| 5 | Discussion | 31 |
| 5.1 | Limites du travail | 31 |
| 5.2 | Perspectives | 31 |
| 5.2.1 | Polytopes intérieurs et récursion maîtrisée | 31 |
| 5.2.2 | Polytopes établis par la physique ou l'apprentissage | 33 |
| 5.3 | Apport personnel | 33 |
| 6 | Conclusion | 35 |
| A | Annexes | 36 |
| | Notations | 36 |
| | Bibliographie | 37 |

1. Introduction

Au fil de la dernière décennie, la montée en flèche des capacités de calcul a fait de l'intelligence artificielle, et plus particulièrement de l'apprentissage par renforcement (*Reinforcement Learning*, RL), un acteur omniprésent : robots d'assistance, véhicules autonomes, systèmes embarqués ou encore jeux de stratégie complexe. Ces agents décisionnels, capables d'interagir avec leur environnement pour maximiser une récompense, promettent des avancées considérables... tout en soulevant une question cruciale : comment garantir que leurs décisions resteront sûres lorsque des vies humaines ou des infrastructures critiques sont en jeu ?

Cette problématique de sûreté est exacerbée par la complexité grandissante des politiques fondées sur des réseaux de neurones profonds, dont le fonctionnement interne demeure largement opaque. Un manque de transparence devient critique dès qu'une décision erronée peut entraîner des conséquences graves : imaginons, par exemple, un module d'atterrissement de fusée ou un drone de livraison qui dévie brusquement de sa trajectoire. Dans de tels contextes, il ne suffit plus d'observer empiriquement les performances : il faut *prouver mathématiquement* que, quels que soient l'état initial et les perturbations rencontrées, l'agent ne franchira jamais les limites de sécurité.

C'est dans ce cadre qu'intervient le programme *DESCARTES* (*DECision-making in Critical URban SysTEMs*), mené au sein du laboratoire *CNRS@CREATE* à Singapour. L'ambition est claire : concevoir des outils d'IA à la fois performants, interprétables et vérifiables pour piloter les systèmes critiques de la ville intelligente — transports, réseaux énergétiques ou flottes de robots urbains. Le stage présenté ici s'inscrit pleinement dans cette dynamique, en s'attachant à expliquer et à valider formellement les décisions prises par des agents RL.

Pour explorer la question dans un cadre contrôlé mais représentatif, je me suis penché sur l'environnement *CartPole* de Gym : une petite voiture se déplace sur un rail et doit maintenir un poteau vertical pendant 500 pas. Bien que plus simple qu'un drone urbain, ce pendule inversé capture l'essence du problème : dynamique non linéaire, horizon temporel long et interactions physiques couplées aux choix de l'agent. Mon étude porte sur la version la plus couramment utilisée — actions et temps discrets — car elle est paradoxalement la plus ardue à certifier. *Ce cadre simplifié offre en outre un point d'entrée idéal pour concevoir des méthodes généralisables à de nombreux autres environnements de contrôle.*

L'objectif est ambitieux : démontrer que, pour tout état initial dans un sous-ensemble continu de l'espace des phases $(x, \dot{x}, \theta, \dot{\theta}) \subset \mathbb{R}^4$, la politique respecte les contraintes de sûreté durant la totalité des 500 étapes. Deux défis majeurs se dégagent : d'une part, traiter un nombre *non dénombrable* de points de départ ; d'autre part, composer à chaque instant avec la double influence de la décision de l'agent et de la dynamique physique de l'environnement.

Face à ces enjeux, la littérature propose un éventail de solutions : extraction d'arbres de décision explicables (VIPER) ou bien apprentissage direct d'arbres obliques différentiables (DTsemNet, SYMPOL). Pour la vérification elle-même, les approches vont de l'abstraction probabiliste (IMDP) aux solveurs SMT, en passant par les certificats de barrière de Lyapunov. Toutes offrent des avancées partielles : chacune repousse une limite — horizon temporel, dimension de l'état ou expressivité de la politique — mais aucune ne fournit encore une garantie globale sur 500 pas dans le cas non linéaire complet de *CartPole*.

Ce travail prolonge les efforts de Benjamin BLOIS, qui avait validé un chemin unique dans une version linéarisée, en apportant deux contributions clés : (i) l'extension de la vérification au modèle non linéaire réaliste ; (ii) le suivi de *toutes* les branches d'un arbre de décision oblique appris via DTsemNet. Grâce à une stratégie d'abstraction-raffinement géométrique, à des découpes adaptatives orientées par la sensibilité des variables et à un parallélisme massif, nous parvenons à certifier la sûreté jusqu'à 125 pas — un premier jalon significatif vers l'horizon complet 500.

En définitive, ce projet s'inscrit dans la quête d'une intelligence artificielle fiable : il démontre qu'en conjuguant politiques interprétables et méthodes formelles, il devient possible de rapprocher la performance du RL de l'exigence de sécurité indispensable aux systèmes critiques de demain.

2. Problématique et contexte scientifique

2.1. Contexte du travail

L'intelligence artificielle, et en particulier l'apprentissage par renforcement (*Reinforcement Learning*, RL), est aujourd'hui omniprésente : robotique, transport autonome, systèmes embarqués, jeux complexes, etc. Ces agents sont capables de prendre des décisions de manière autonome dans des environnements variés.

Cependant, leur complexité croissante — notamment lorsqu'ils reposent sur des réseaux de neurones profonds — rend leurs comportements difficiles à comprendre et à expliquer. Ce manque de transparence pose un problème majeur dès lors qu'une mauvaise décision peut entraîner des conséquences graves (par exemple, dans le cas d'un module d'atterrissement de fusée). Il devient donc crucial de vérifier mathématiquement que les décisions prises par un agent sont sûres, quelles que soient les états initiaux du système.

C'est dans ce cadre que s'inscrit mon stage, réalisé au sein du laboratoire CNRS@CREATE, filiale du CNRS implantée à Singapour. Ce centre de recherche interdisciplinaire accueille plusieurs projets stratégiques liés à la modélisation intelligente des systèmes urbains, dont le programme DESCARTES.

Le projet DESCARTES, acronyme de *DECision-making in Critical URban SysTEMs*, a pour ambition de développer des outils d'intelligence artificielle sûrs, interprétables et performants pour piloter des systèmes critiques au sein des villes intelligentes du futur. Ces systèmes incluent notamment des infrastructures de transport, des réseaux énergétiques, ou encore des agents autonomes tels que des drones, des véhicules ou des robots urbains.

Une attention particulière est ainsi portée à la prise de décision automatique dans des environnements urbains complexes, où les agents doivent interagir en toute sécurité avec des humains et des infrastructures. Par exemple, un drone autonome chargé de livrer un colis ou de surveiller une zone urbaine doit être capable de naviguer sans heurter un bâtiment ni mettre en danger un passant. Cela nécessite des garanties fortes sur les politiques de décision mises en œuvre.

Le travail effectué s'inscrit précisément dans cette problématique : il vise à expliquer et à vérifier les décisions prises par des agents intelligents — en l'occurrence, par des modèles d'apprentissage par renforcement — afin de s'assurer que leur comportement demeure sûr, quelles que soient les conditions initiales.

2.2. Présentation du problème

2.2.1. Présentation de *Cartpole*

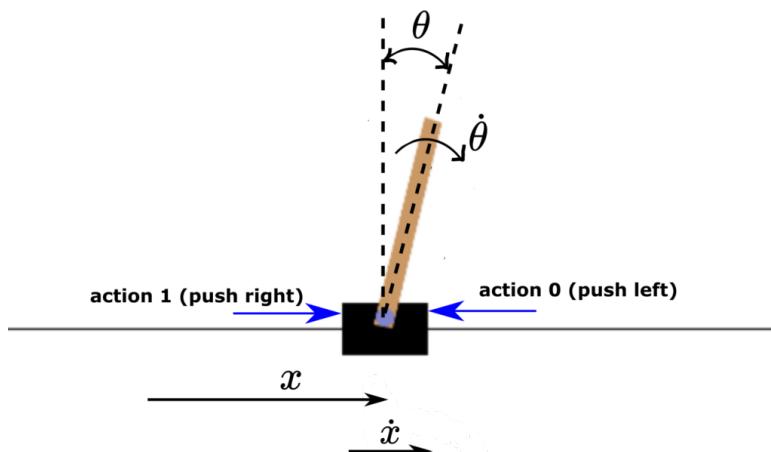


FIGURE 2.1. – Illustration de l'environnement CartPole-v1 de Gym

Plus précisément, le stage s'inscrit dans un problème un peu plus simple et plus formel que celui des drones à proprement parler : le problème CartPole-v1 de l'environnement Gym. Il s'agit d'un environnement 2D composé

d'une petite voiture se déplaçant sur un axe, et devant stabiliser un poteau fixé dessus en position verticale. L'agent doit maintenir le poteau dans une plage angulaire proche de la verticale, tout en gardant la voiture entre deux murs, et cela pendant 500 étapes consécutives.

L'environnement peut se décliner sous différentes formes :

- **Actions discrètes vs continues** : pour les actions discrètes, l'agent applique une force d'une intensité fixée ; pour les actions continues, l'agent choisit directement l'intensité de la poussée.
- **Temps discret vs continu** : en temps discret, une action est appliquée à chaque intervalle de temps fixe, tandis qu'en temps continu, on choisit le moment précis où l'action est appliquée.

Dans notre cas, nous traitons uniquement le cas **discret en actions et en temps**, qui correspond à la version la plus utilisée (et aussi la plus délicate à certifier) de CartPole.

Objectifs : Ainsi, le but est de prouver que pour un ensemble de situation (ie pour un sous-ensemble de \mathbb{R}^4 , espace des phases $(x, \dot{x}, \theta, \dot{\theta})$), l'agent respecte l'ensemble des contraintes sur 500 pas.

Les principaux enjeux résident dans la nécessité de certifier un ensemble non dénombrable de points de départ, ainsi que dans la grande variabilité des trajectoires possibles selon le comportement de l'agent. En effet, à chaque étape de la trajectoire deux éléments doivent être pris en compte : la décision de l'agent (politique RL), et l'évolution donnée par la physique de l'environnement (f_{physique}).

2.2.2. L'apprentissage par renforcement

L'environnement CartPole-v1 peut être formalisé comme un *processus de décision markovien* (MDP) défini par le quintuplet

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, f_{\text{phys}}, r, \gamma \rangle.$$

- **États** : $\mathcal{S} \subseteq \mathbb{R}^4$ correspond à l'espace des phases $s = (x, \dot{x}, \theta, \dot{\theta})$ du chariot-pendule.
- **Actions** : $\mathcal{A} = \{-F, +F\}$ représente l'application d'une force constante vers la gauche ou vers la droite ¹.
- **Dynamique** : $f_{\text{phys}} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ décrit l'évolution non linéaire du système :

$$s_{t+1} = f_{\text{phys}}(s_t, a_t) = s_t + \Delta t \phi(s_t, a_t),$$

où g est obtenu par les équations d'Euler–Lagrange du pendule inversé.

- **Récompense** : $r(s_t, a_t) = 1$ tant que $|x_t| \leq x_{\max}$ et $|\theta_t| \leq \theta_{\max}$, puis 0 si l'épisode termine.
- **Facteur d'actualisation** : $\gamma \in (0, 1]$; on fixe souvent $\gamma = 0.99$ afin de privilégier l'horizon long (jusqu'à $T = 500$ pas).

Objectif d'apprentissage. Une *politique* $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ — où $\Delta(\mathcal{A})$ désigne ici l'ensemble des distributions de probabilité sur les deux actions discrètes $\{-F, +F\}$, autrement dit les probabilités p telles que $\pi(s) = (p, 1 - p)$ pour chaque état s — maximise le retour espéré des récompenses

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \right],$$

sous la dynamique $s_{t+1} \sim f_{\text{phys}}(s_t, a_t)$. L'*algorithme d'apprentissage par renforcement* consiste à estimer un optimal $\pi^* = \arg \max_\pi J(\pi)$.

De nombreuses méthodes permettent aujourd'hui d'estimer cette politique, parmi lesquelles le *Temporal Difference Learning* [10] et le *Q-learning* [11]. L'approche dominante actuelle, le *Deep Reinforcement Learning*, repose sur l'usage de réseaux de neurones, rendant la vérification formelle des politiques particulièrement difficile.

Représentation de la politique. Dans le cadre du programme DESCARTES, Pandia *et al.* [2] ont introduit **DTsemNet**, un réseau de neurones spécialement conçu pour apprendre un *arbre de décision oblique* sémantiquement équivalent. Concrètement, l'entraînement s'effectue par descente de gradient — comme pour un réseau standard —, mais la structure induite reste celle d'un arbre binaire T dont chaque nœud applique un test affine

$$w^\top s \leq b,$$

1. Dans la version continue, on aurait $\mathcal{A} = [-F_{\max}, F_{\max}]$, mais nous restons ici dans le cas discret.

et dont chaque feuille assigne une action $a \in \mathcal{A}$. Cette hybridation combine :

- (i) la *performance* d'un modèle neuronal grâce à l'optimisation continue ;
- (ii) l'*interprétabilité* et la *vérifiabilité* d'un arbre, puisque T partitionne l'espace d'état en un nombre fini (≤ 64 dans les expériences) de *polytopes convexes*

$$P_k = \{s \in \mathcal{S} \mid A_k s \leq b_k\},$$

sur lesquels l'action est constante.

Géométriquement, chaque nœud ajoute un demi-espace, et la descente récursive définit un polytope de plus en plus restreint ; la politique

$$\pi_T(s) = a_k \quad \text{si } s \in P_k$$

se lit donc comme une table de décision par région. Cette description est idéale pour la *vérification formelle* : on peut propager analytiquement (ou par sur-approximation) chaque polytope P_k sous la dynamique f_{phys} et démontrer que toutes les trajectoires respectent les contraintes de sûreté.

La figure 2.2 illustre la partition induite par T dans différentes projections (x, \dot{x}) , (x, θ) , etc., tandis que la figure 2.3 montre la structure complète de l'arbre, où les flèches vertes (et rouges) indiquent respectivement les branches TRUE (FALSE) d'un test.

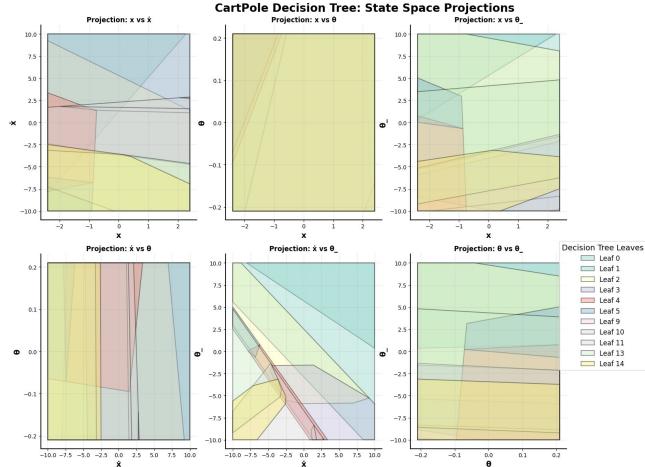


FIGURE 2.2. – Projection de la politique oblique sur l'espace d'état. Chaque couleur correspond à une feuille (et donc à une action de $\{-F, +F\}$).

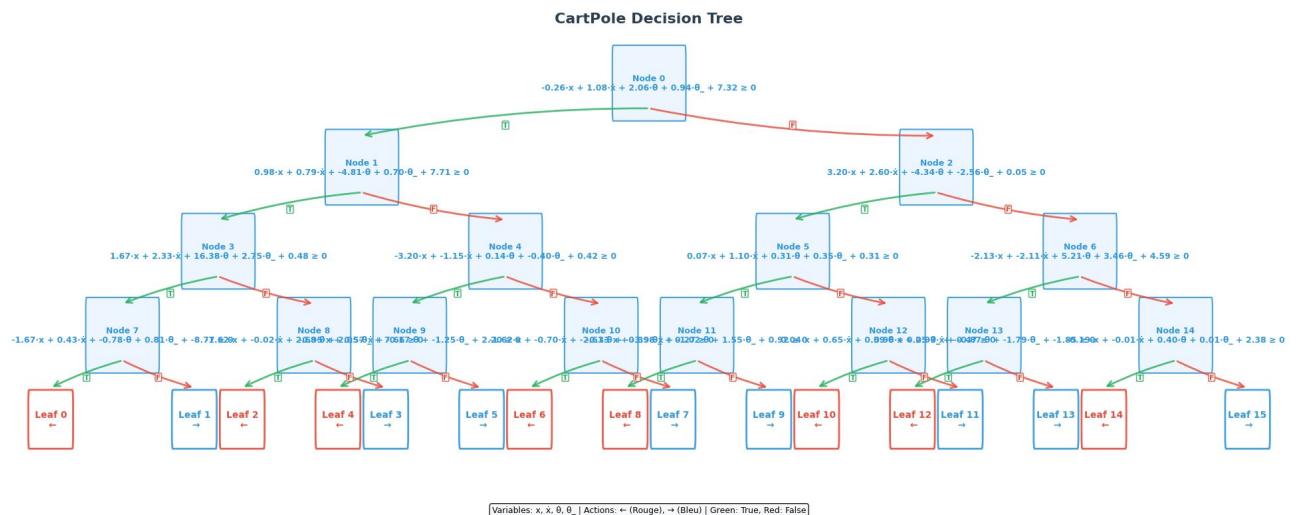


FIGURE 2.3. – Structure binaire complète de l'arbre. Les inégalités sont affichées dans chaque nœud ; les feuilles bleues prescrivent l'action \rightarrow , les rouges l'action \leftarrow .

En résumé, l'encodage direct d'une politique RL sous la forme d'un arbre de décision oblique offre un compromis

robuste : *puissance statistique* à l'entraînement, *lisibilité* et *compatibilité* avec les méthodes formelles que nous exploitez dans les sections suivantes.

Ainsi formulé, le problème de *CartPole* sert de banc d'essai aux différentes méthodes ; celles-ci pourront ensuite se transposer à d'autres environnements continus ou discrets présentant la même structure MDP.

2.3. État de l'art et analyse critique

2.3.1. Vers des politiques *RL* interprétables

Ainsi que nous l'avons établi, l'*interprétabilité* constitue un atout décisif : elle facilite la compréhension des modèles, l'explicabilité des résultats et, surtout, l'application de méthodes de vérification rigoureuses. C'est dans cette perspective qu'a été développé **DTsemNet** [2], dont l'objectif est de condenser la stratégie d'un agent dans un arbre de décision oblique à la fois compact et géométriquement explicite.

D'autres approches poursuivent un objectif similaire en construisant directement des arbres interprétables au cours de l'entraînement de la politique :

- **SYMPOL** [3] démontre l'entraînement *end-to-end* d'arbres axis-alignés via PPO, atteignant (et parfois dépassant) les performances des approches neuronales classiques.
- **MoET** (*Mixture of Expert Trees*) [4] exploite plusieurs petits arbres spécialisés, sélectionnés dynamiquement par une fonction de *gating* linéaire, combinant ainsi interprétabilité locale et couverture globale de l'espace d'état.

Par ailleurs, **VIPER** [1] a introduit une approche différente, consistant à extraire, *a posteriori*, un arbre axis-aligné à partir d'une politique neuronale déjà entraînée, ouvrant la voie à une interprétation symbolique du comportement de l'agent.

Toutefois, si ces représentations arborées clarifient la logique de la politique, elles ne suffisent pas à en garantir la *sûreté*. Il reste en effet à démontrer, de manière formelle, que l'ensemble des trajectoires induites respecte les contraintes du système physique. La sous-section suivante présente les approches existantes qui visent à établir de telles garanties.

2.3.2. Méthodes de vérification formelle

Il est désormais possible (dans certains cas) d'expliquer les décisions prises à une étape donnée par une politique. En revanche, expliquer une séquence de décisions sur un horizon de 500 étapes reste beaucoup plus complexe. En effet, à chaque étape, la dynamique physique du système modifie l'espace des phases suivi, ce qui transforme également les régions de décision correspondantes. Autrement dit, si l'on peut analyser et comprendre une décision isolée, il n'est pas encore possible, à ce jour, de suivre et d'expliquer l'évolution d'un ensemble d'états tout au long des 500 étapes, car la dynamique du système altère trop fortement les représentations de décision.

Les sous-sections ci-dessous détaillent, pour chaque famille de méthodes, le principe général, la mise en œuvre typique sur *CartPole-v1*, les résultats obtenus, ainsi que leurs principales limites.

a) Vérification bornée par SMT : VIPER

La méthode VIPER consiste à distiller la politique neuronale profonde en un petit arbre de décision (DT) grâce à l'apprentissage par imitation, puis à linéariser localement la dynamique du pendule pour l'encoder — avec l'arbre — dans un problème SMT résolu par Z3[1]. Un problème SMT (Satisfiability Modulo Theories) consiste à vérifier si un ensemble de contraintes logiques et mathématiques admet une solution ; Z3 est un solveur capable de démontrer automatiquement si, pour toute configuration initiale admissible, les conditions de sûreté sont respectées. Sur *CartPole-v1*, cette combinaison fournit une preuve que la perche reste dans les bornes pendant les 10 premiers pas pour tout état initial dans $[-0.05, 0.05]^4$. L'approche est exacte et produit des contre-exemples lorsqu'une propriété échoue, mais elle reste limitée à de courts horizons, dépend fortement de l'hypothèse d'angle petit et voit la taille de l'arbre — donc la complexité SMT — croître exponentiellement avec la dimension de l'état.

b) Analyse géométrique et certificats de barrière

Cette approche repose sur la construction d'une fonction appelée *certificat de barrière* (souvent de type Lyapunov-barrière) $B(x)$, conçue pour décroître (ou ne pas augmenter) le long des trajectoires du système. Si une telle fonction peut être apprise et vérifiée, elle constitue une preuve que les trajectoires restent confinées dans une zone sûre \mathcal{S} , et ce sans limite de temps — ce qui permet théoriquement de garantir la sûreté sur un horizon infini (e.g. [6]). En pratique, la fonction barrière est généralement entraînée en même temps que la politique, via un objectif d'optimisation contraint.

Plus récemment, ces idées ont été adaptées au RL profond : Luo & Ma [7] co-entraînent la politique, le modèle dynamique et un certificat de barrière afin d'obtenir *zéro violation* pendant l'entraînement, tandis que Mandal *et al.* [8] apprennent puis *vérifient formellement* des certificats Lyapunov–barrière pour des contrôleurs d'apprentissage par renforcement profond (DRL), montrant la faisabilité de garanties formelles à horizon infini dans des cadres de complexité maîtrisée. Du côté des systèmes hybrides et du contrôle, Zhao *et al.* [9] proposent de *synthétiser* des certificats de barrière à l'aide de réseaux de neurones, illustrant que l'apprentissage de telles fonctions peut être automatisé et ensuite validé.

Cette méthode présente plusieurs avantages, notamment la possibilité de garantir la sûreté sans dérouler explicitement les trajectoires (on raisonne sur une fonction scalaire B plutôt que sur toutes les trajectoires possibles). Néanmoins, elle repose sur des hypothèses fortes (stabilisabilité, connaissance ou bonne approximation des dynamiques, régularité du certificat) et sur la réussite de l'apprentissage d'un certificat globalement valide, ce qui reste difficile en pratique. De plus, les solveurs actuels utilisés pour vérifier ces certificats (quand ils sont paramétrés par des réseaux de neurones) souffrent encore de problèmes de passage à l'échelle pour des systèmes de grande dimension ou des environnements RL complexes. Actuellement, il n'existe pas encore de preuve publiée et complète, basée sur des certificats de barrière, couvrant l'environnement *CartPole* dans toute sa plage d'états et sur l'horizon total ; les résultats restent soit partiels, soit limités à des sous-ensembles d'états ou à des horizons bornés, ou encore s'appuient sur des architectures de politiques spécialement conçues pour être plus facilement vérifiables.

c) Propagation continue dans un arbre axis-aligné

Schilling *et al.* proposent de propager directement la solution analytique de l'ODE de *CartPole* à travers chaque feuille d'un arbre de décision axis-aligné, évitant ainsi la discréétisation temporelle et l'accumulation d'erreurs [5]. Conceptuellement, la méthode peut fournir des garanties globales avec moins de sur-approximation ; en pratique, elle n'a pour l'instant été démontrée que sur des arbres peu profonds. Les auteurs sont néanmoins parvenus à prouver, pour un certain arbre utilisant des actions discrètes en temps discret, et uniquement pour le problème de *Cartpole* qui ne constraint pas la dimension x , que le système restait sûr sur un horizon infini.

2.3.3. Défis ouverts

- **Scalabilité horizon 500.** Étendre les garanties SMT au pas complet de *CartPole-v1* (500 steps) reste coûteux. L'intégration de mécaniques mixtes (arbres compacts + invariants de Lyapunov) est une piste active.
- **Compromis expressivité / vérifiabilité.** Les arbres obliques ou mélanges d'arbres améliorent la performance, mais gonflent l'expression logique, rapprochant parfois la complexité de celle d'un réseau dense.
- **Linéarité.** La majorité des résultats se placent dans un cadre linéaire afin de pouvoir résoudre les systèmes de manière plus simple et plus rapide, mais le cas général de *CartPole* est non linéaire.

2.4. Travaux réalisés précédemment

Mon prédecesseur, Benjamin Blois, a travaillé dans le cadre du projet DESCARTES sur la vérification de politiques d'apprentissage par renforcement obtenues via l'outil DTsemNet. Ce modèle permet de générer des politiques sous forme d'arbres de décision obliques, c'est-à-dire définies par des inégalités linéaires générales (et non simplement axis-aligned), induisant une partition convexe de l'espace des états.

Son approche consistait à vérifier la sûreté de ces politiques dans l'environnement *CartPole*, en s'assurant qu'à chaque étape, les états générés par la politique restaient dans une zone considérée comme sûre. Pour cela, il a développé une méthode d'abstraction-raffinement fondée sur la représentation géométrique des états atteignables

à l'aide de polytopes convexes. À chaque étape, ces volumes étaient approximés par des formes simples (telles que des octogones) pour limiter la complexité du calcul.

Cependant, cette vérification a été réalisée uniquement dans un cadre linéaire, en suivant une seule trajectoire possible de la politique : à chaque étape, un unique sous-volume était sélectionné, correspondant à une seule branche de l'arbre de décision. Autrement dit, au lieu d'explorer l'ensemble des bifurcations possibles dictées par la politique, il suivait un seul chemin à travers l'arbre — puis y appliquait la physique.

Cette stratégie a permis d'atteindre une profondeur de 500 étapes dans l'arbre dans le cas linéaire, mais ne permettait pas d'apporter une garantie globale sur l'ensemble des comportements possibles de la politique. Elle constituait néanmoins une certaine avancée par rapport aux méthodes antérieures (comme VIPER), qui ne permettaient de vérifier que les toutes premières étapes (mais pour l'intégralité des branches).

2.5. Contributions

Mon travail s'inscrit dans la continuité de cette approche, mais se distingue par deux contributions majeures : d'une part, l'analyse est étendue au cas non linéaire, permettant de représenter des comportements plus réalistes et complexes ; d'autre part, par le suivi de l'ensemble des branches possibles de la politique, afin de fournir une garantie globale sur l'ensemble des trajectoires qu'un agent peut emprunter, jusqu'à une **profondeur de 125 en employant de nouvelles techniques**.

2.5.1. Déroulement de la recherche

Semaines 1–2 : Ré-exécution et refactorisation du cas linéaire

- Ré-import des scripts de [12], mise en place d'une architecture orientée-objet.
- Validation des résultats obtenus précédemment.

Semaines 2–5 : Implémentation du cas non linéaire sur un chemin unique

- Ré-écriture fidèle de la dynamique CartPole pour éliminer les divergences d'implantation.
- Encadrement direct de l'image d'un polytope (cf. Section 3.3.2).

Semaines 5–10 : Extension au cas général : suivi exhaustif de tous les chemins

- Mise en place de la gestion globale (gestion de la croissance exponentielle en 2^n), de la clusterisation.
- Introduction du parallélisme (*threading*) pour la résolution simultanée de polytopes indépendants.

Semaines 10–16 : Exploration

- Détection automatique des ré-apparitions d'ensembles (Section 3.4.1).
- Ajustement dynamique du nombre de polytopes conservés.
- Approximation intérieure des clusters par un polytope convexe minimal et soustraction aux polytopes suivis.
- Parcourt d'arbre.
- Découpe adaptative le long des directions sensibles (angles du pendule et vitesse du chariot).

3. Travail réalisé

L'ensemble du travail a été guidé par une démarche scientifique agile et s'est déroulé en trois phases successives : (i) reproduction et consolidation du cas linéaire sur une trajectoire unique, (ii) généralisation au cas non linéaire, puis (iii) extension à une abstraction couvrant l'ensemble des chemins.

3.1. Préambule : Représentation des états et objectifs

À chaque instant, l'état du système CartPole est décrit par le vecteur

$$\mathbf{X} = (x, \dot{x}, \theta, \dot{\theta})^T \in \mathbb{R}^4$$

et l'action discrète $a_{\mathbf{X}} \in \{0, 1\}$ modifie la dynamique non linéaire $\dot{\mathbf{X}} = f(\mathbf{X}, a_{\mathbf{X}})$. Ainsi, ensemble d'état dans l'espace des phases est représenté par un ensemble de \mathbb{R}^4 . De ce fait, dans toute la suite, nous notons

$$\mathcal{P} = \{x \in \mathbb{R}^4 \mid Ax \leq b\}$$

un polytope convexe donné par son écriture *H-représentation*. Notre objectif est d'obtenir un sur-encadrement (abstraction) $\mathcal{O}(\mathcal{I}(\mathcal{P}))$ de l'image d'un polytope par la dynamique exacte en un pas de temps Δt , et de vérifier si pour ce pas de temps cet encadrement est dans les bornes de sûreté (*). En faisant cela au cours de 500 étapes nous pourrons affirmer que le polytope initial a été inclus dans chacun des \mathcal{P}_n aux étapes correspondantes et donc dire que celui-ci est sûr pendant les 500 étapes.

De plus, la politique divise en moyenne chaque polytope en deux (soit 2 polytopes générés en moyenne par polytope source). Dans un premier temps, nous ne suivons qu'un seul des deux polytopes issus de chaque division — c'est-à-dire un choix de la politique à chaque étape — et nous procédons à la vérification (*). Cependant, dans un second temps, il devient nécessaire de suivre l'ensemble des polytopes issus de l'intersection avec la politique à chaque étape : il faut alors encadrer des ensembles de polytopes à chaque étape, et vérifier qu'ils restent dans les bornes de sûreté tout comme pour (*). Cela permet d'affirmer que le polytope initial est inclus dans les encadrements définis à l'étape n , pour tout $n \in \{0, \dots, 500\}$, et donc de conclure à sa sûreté dans le cas général.

3.2. Phase 1 – Reproduction du cas linéaire sur un seul chemin

3.2.1. Hypothèses et méthodes

L'objectif initial était de réimplémenter l'algorithme précédent ([12]) afin de disposer d'une base de référence fiable. Ainsi, nous considérons l'agent RL sous une forme *DT oblique* entraîné dans la dynamique *exacte* de CartPole, mais avec une vérification d'un chemin qui s'appuie sur l'*hypothèse linéaire* (H_{LIN}) : à chaque étape, l'image d'un point est l'image faite par l'approximation linéaire.

Hypothèse ($H_{LIN-PHYS}$) : on suppose que la dynamique réelle est proche de sa version linéarisée :

$$f_{\text{physique}}(P) = f_{\text{physique}}^{\text{lin}}(P) \tag{3.1}$$

Conséquence 1 $\mathcal{I}(\mathcal{P})$: Etant donné que $f_{\text{physique}}^{\text{lin}}$ est une application linéaire, on a :

$$\mathcal{I}(\mathcal{P}) = f_{\text{physique}}^{\text{lin}}(\mathcal{P}) = \text{conv}(\{f_{\text{physique}}^{\text{lin}}(v) \mid v \in \mathcal{S}(\mathcal{P})\}) \tag{3.2}$$

Conséquence 2 $\mathcal{A}(\mathcal{P})$: Pour tout $x \in \mathcal{I}(P)$, il existe des coefficients $\lambda_j \geq 0$, $\sum_j \lambda_j = 1$ tels que :

$$x = \sum_j \lambda_j \text{ip}_j, \quad \text{avec } \text{ip}_j \in \mathcal{S}(\mathcal{I}(P)) \tag{3.3}$$

Par linéarité, on a donc :

$$x = f_{\text{physique}}^{\text{lin}} \left(\sum_j \lambda_j p_j \right), \quad \text{avec } \sum_j \lambda_j p_j \in P \text{ et } p_j \in \mathcal{S}(P) \quad (3.4)$$

Cette linéarisation introduit un écart systématique entre la dynamique exacte et son approximation qui est non négligeable sur les bords du domaine.

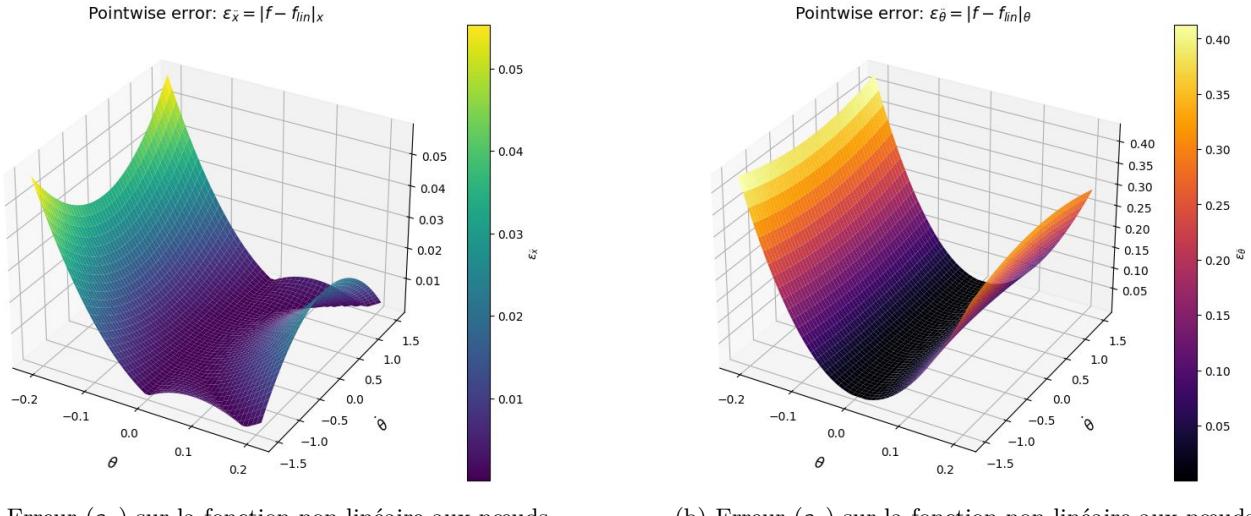


FIGURE 3.1. – Erreur sur la linéarisation de f_{physique}

Ordre de grandeur de ε . Les deux graphes ci-dessus donnent un ordre de grandeur de $\varepsilon = |f_{\text{physique}} - f_{\text{physique}}^{\text{lin}}|$ aux noeuds des pavés. L'erreur est très faible autour de $(\theta, \dot{\theta}) = (0, 0)$ (de l'ordre de 10^{-5}), mais elle peut atteindre $\approx 5 \times 10^{-2}$ sur les bords ; la non-linéarité y devient alors non négligeable et doit être prise en compte (d'autant plus qu'ce sont les zones les plus critiques).

Encadrement par $\mathcal{O}(\mathcal{I}(P))$ L'ensemble $\mathcal{I}(P)$ contient un trop grand nombre d'équations, qui croît rapidement au fil des pas de temps. Pour limiter cette explosion, on encadre $\mathcal{I}(P)$ par $\mathcal{O}(\mathcal{I}(P))$, une sur-approximation octogonale construite à partir de $\mathcal{I}(P)$, et qui contient donc beaucoup moins d'équations.

Cette approximation permet de réduire drastiquement la complexité, au prix toutefois d'une perte de précision. Les erreurs introduites sont contrôlées par notre algorithme de *raffinement-backtracking*.

Raffinement adaptatif et *backtracking* en dynamique linéaire Une fois l'image octogonale $\mathcal{O}(\mathcal{I}(P))$ calculée, nous vérifions sa sûreté en évaluant la trajectoire des *sommets* du polytope. Si tous ces sommets satisfont les contraintes, l'ensemble est réputé sûr au pas considéré. Dans le cas contraire, chaque sommet contre-exemple est ajouté aux *points problématiques*¹ de l'étape courante. Chaque point problématique est traité individuellement :

- (a) **Ajout dirigé d'une contrainte.** Nous sélectionnons, dans l'image exacte $\mathcal{I}(P)$, l'hyperplan $h(s) \leq 0$ qui enlève le mieux le point problématique. Cette inégalité est insérée dans l'ensemble d'équations : l'octogone se resserre et le point disparaît si la marge d'exclusion est suffisante.
- (b) **Backtracking si nécessaire.** Lorsque le point ne peut pas être éliminé avec une marge satisfaisante, nous remontons d'un pas : l'antécédent exact du projeté sur $\mathcal{I}(P)$ est ajouté aux points problématiques de l'étape précédente, puis on recommence l'étape précédente. Cette opération empêche le polytope affiné de régénérer, aux pas suivants, le point faux.

Ce mécanisme «raffinement + backtracking» garantit qu'une abstraction trop grossière est systématiquement resserrée jusqu'à ce que toutes les trajectoires respectent les contraintes, pourvu que le polytope initial soit lui-même sûr.

1. «Point problématique» désigne un point «faux»[†] trop «proche».

2. [†] Un «point faux» est un point dont la trajectoire ne respecte pas les contraintes.

3.2.2. Présentation d'une étape

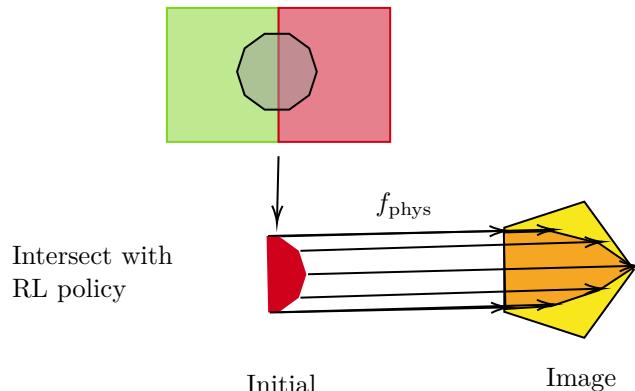


FIGURE 3.2. – Présentation d'un pas de l'algorithme dans le cas linéaire en suivant un seul chemin

Ainsi, l'algorithme précédent consiste à chaque étape à :

- intersecer le polytope courant avec la politique de l'agent de renforcement (RL), puis à choisir l'un des morceaux à suivre (ici, le morceau rouge)
- calculer l'image de chacun de ses sommets par la physique linéaire
- l'encadrer par un octogone (directions prédéfinis) puis à raffiner cet octogone
 - Si l'on parvient à raffiner cet octogone de façon à exclure chacun des points problématiques, on passe à l'étape suivante.
 - Sinon, on effectue un *backtrack*, c'est-à-dire que l'on revient à l'étape précédente en déclarant l'antécédent (du point problématique) comme problématique à l'étape précédente, puis on repart de l'étape précédente.

3.3. Phase 2 – Passage au cas non linéaire, suivi d'un seul chemin

3.3.1. Présentation des changements théoriques

La seconde étape a consisté à lever l'approximation linéaire, en travaillant directement avec la dynamique exacte de *CartPole*, afin d'éviter les divergences entre la physique exacte et celle approchée.

Encadrement direct de $\mathcal{O}(\mathcal{I}(P))$ Plutôt que d'utiliser les images des sommets via la physique linéaire, puis de calculer $\mathcal{O}(\mathcal{I}(P))$, le nouvel algorithme procède différemment.

On commence par calculer une borne sur la dérivée seconde, fournie par la physique. Ensuite, on optimise dans des directions arbitraires (directions alignées avec les axes et combinaisons linéaires simples), en utilisant l'évolution sur un pas de temps. Cela permet de construire directement $\mathcal{O}(\mathcal{I}(P))$, sans avoir à calculer l'image de points arbitraires, et donc d'obtenir un encadrement direct de toute l'image du polytope, uniquement à partir de P .

Les bornes sont obtenues par min-max des dérivées secondes, puis résolues sous forme de programmes linéaires (à l'aide de solveurs linéaires), plus un raffinement si nécessaire.

Calcul d'antécédents Contrairement au cas linéaire, l'opération \mathcal{I} n'est ni linéaire ni bijective. Nous utilisons donc une stratégie d'*encadrement d'antécédents*. À partir d'un point $y \in \mathcal{O}(\mathcal{I}(P))$, on cherche un polytope convexe $Q \subseteq P$, le plus petit possible, tel que $\mathcal{O}(\mathcal{I}(Q)) \supseteq \{y\}$.

3.3.2. Encadrement octogonal (cas non-linéaire)

Première version : borne globale sur la dérivée seconde

Soit $\mathbf{X} = (x, \dot{x}, \theta, \dot{\theta})^T \in \mathbb{R}^4$ un état appartenant à un polytope \mathcal{P} sur lequel l'action de politique est constante (ie $\forall \mathbf{Y} \in \mathcal{P}, \pi(Y) = a_{\mathcal{P}}$). Les paramètres physiques sont : m_c la masse du chariot, m_p celle de la tige, ℓ sa demi-longueur, et g l'accélération gravitationnelle.

Pour un pas de temps Δt , on a :

$$\Phi(\mathbf{X}, a_{\mathcal{P}}) = \begin{cases} \ddot{x} = \frac{F_{\mathcal{P}} + m_p \sin \theta (\ell \dot{\theta}^2 + g \cos \theta)}{m_c + m_p \sin^2 \theta}, \\ \ddot{\theta} = \frac{-F_{\mathcal{P}} \cos \theta - m_p \ell \dot{\theta}^2 \cos \theta \sin \theta - (m_c + m_p) g \sin \theta}{\ell (m_c + m_p \sin^2 \theta)} \end{cases} \quad (\text{Équations dynamiques})$$

$$\begin{cases} x_{n+1} = x_n + \dot{x}_n \Delta t, \\ \dot{x}_{n+1} = \dot{x}_n + \ddot{x}_n \Delta t, \\ \theta_{n+1} = \theta_n + \dot{\theta}_n \Delta t, \\ \dot{\theta}_{n+1} = \dot{\theta}_n + \ddot{\theta}_n \Delta t \end{cases} \quad (\text{Mise à jour temporelle - méthode d'Euler}) \quad (\ddot{x}_n, \ddot{\theta}_n) = \Phi(\mathbf{X}_n, a_{\mathcal{P}})$$

avec l'action discrète $F_{\mathcal{P}} = (2 \cdot a_{\mathcal{P}} - 1) \cdot F_0 \in \{\pm F_0\}$.

Ainsi, à $F_{\mathcal{P}}$ fixé sur l'ensemble d'un polytope \mathcal{P} , en prenant le max et le min de $\Phi(X, a_{\mathcal{P}})$ pour tout $X \in \mathcal{P}$, on obtient des bornes $\varepsilon_{\ddot{x}}^{\min}, \varepsilon_{\ddot{x}}^{\max}$ et $\varepsilon_{\ddot{\theta}}^{\min}, \varepsilon_{\ddot{\theta}}^{\max}$ de \mathbf{X} sur \mathcal{P}

$$\varepsilon_{\ddot{x}, \mathcal{P}}^{\max} = \max_{\mathcal{P}} |\Phi(\mathbf{X}, a_{\mathcal{P}})_{\ddot{x}}|$$

Construction de l'encadrement.

1. **Extension de l'espace.** On enrichit l'espace d'état par les accélérations $(\ddot{x}, \ddot{\theta})$, limitées grâce à un programme linéaire aux intervalles $\ddot{x} \in [\varepsilon_{\ddot{x}}^{\min}, \varepsilon_{\ddot{x}}^{\max}]$ et $\ddot{\theta} \in [\varepsilon_{\ddot{\theta}}^{\min}, \varepsilon_{\ddot{\theta}}^{\max}]$. Le polytope initial $\mathcal{P} \subset \mathbb{R}^4$ est ainsi plongé dans \mathbb{R}^6 , $\mathcal{P}_{\text{ext}} \subset \mathbb{R}^6$.
2. **Approximation octogonale.** Soit n la direction que l'on souhaite optimiser ($n \in \{\pm e_i, e_i \pm e_j \mid i < j\}$, normalisé).

Soient

$$\alpha = \begin{pmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \end{pmatrix}$$

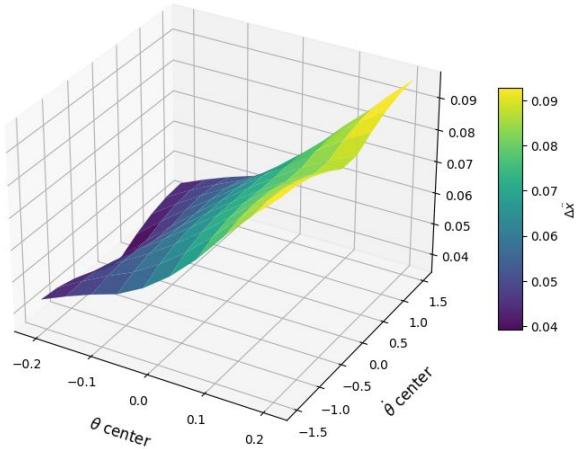
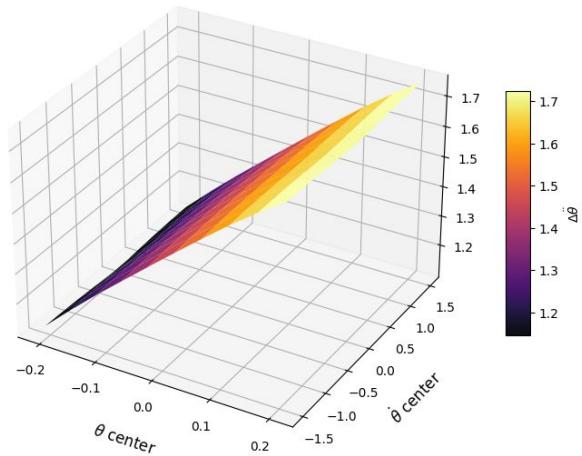
On résout

$$m = \min_{z \in \mathcal{P}_{\text{ext}}} \langle n \cdot \alpha, z \rangle, \quad M = \max_{z \in \mathcal{P}_{\text{ext}}} \langle n \cdot \alpha, z \rangle$$

Ces contraintes $\langle n, \mathbf{X}' \rangle \leq M$, $-\langle n, \mathbf{X}' \rangle \leq -m$. définissent des hyperplans, en prenant 16 couples on obtient un octogone $\mathcal{O}(\mathcal{I}(\mathcal{P})) \subset \mathbb{R}^4$, sur-approximant l'image exacte de \mathcal{P} .

Cette méthode est peu coûteuse (16 programmes linéaires), mais l'encadrement peut rester lâche lorsque \mathcal{P} possède à certains endroits de fortes accélérations dans sa dynamique, mais également dans des directions spécifiques qui n'ont pas été optimisées.

Encadrement direct avec la physique non linéaire. D'après 3.3, on constate que l'encadrement direct de la fonction non linéaire est assez lâche : de l'ordre de 10^{-1} pour \ddot{x} et de 1.7 pour $\ddot{\theta}$ pour des pavés de taille 0.04×0.3 en $(\theta, \dot{\theta})$. Avec un pas $\Delta\tau = 10^{-2}$, on obtient des valeurs respectives de 10^{-3} et 1.7×10^{-2} , bien trop élevées pour notre problème ; un tel encadrement n'est donc pas satisfaisant.

Variation of $\Delta\ddot{x}$ over local intervals(a) Ordre de grandeur de l'encadrement du résidu
 $\Delta\ddot{x} = \Delta\epsilon_{\ddot{x}}$ sur chacun des pavésVariation of $\Delta\ddot{\theta}$ over local intervals(b) Ordre de grandeur de l'encadrement du résidu
 $\Delta\ddot{\theta} = \Delta\epsilon_{\ddot{\theta}}$ sur chacun des pavésFIGURE 3.3. – Ordre de grandeur obtenu par encadrement direct de la dérivée seconde f_{physique} **Version améliorée : soustraction de la composante linéaire**

Composante Linéaire On peut décomposer

$$\Phi(\mathbf{X}, a_{\mathcal{P}}) = B(a_{\mathcal{P}})\mathbf{X} + c(a_{\mathcal{P}}) + g(\mathbf{X}, a_{\mathcal{P}})$$

où $B(a)$ est la composante linéaire, $c(a)$ la partie affine due à l'action, et g le résidu non linéaire.

$$B(a_{\mathcal{P}}) = B = \begin{pmatrix} 0 & 0 & \frac{m_p g}{m_c} & 0 \\ 0 & 0 & \frac{(m_c + m_p)g}{m_c \ell} & 0 \end{pmatrix}, \quad c(a_{\mathcal{P}}) = \begin{pmatrix} F_0 (-1)^{a_{\mathcal{P}}} \\ \frac{m_c}{F_0} (-1)^{a_{\mathcal{P}}} \\ \frac{F_0}{m_c \ell} (-1)^{a_{\mathcal{P}}} \end{pmatrix}$$

$$B_{\text{plong}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{m_p g}{m_c} & 0 & 1 \\ 0 & 0 & \frac{(m_c + m_p)g}{m_c \ell} & 0 & 0 \end{pmatrix}, \quad c_{\text{plong}}(a_{\mathcal{P}}) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{F_0}{m_c} (-1)^{a_{\mathcal{P}}} \\ \frac{F_0}{m_c \ell} (-1)^{a_{\mathcal{P}}} \end{pmatrix}$$

Bornes Ainsi au lieu de prendre $\varepsilon_{\ddot{x}, \mathcal{P}}^{\max} = \max_{\mathcal{P}} |\Phi(\mathbf{X}, a_{\mathcal{P}})|$, on prend $\epsilon_{\ddot{x}, \mathcal{P}}^{\max} = \max_{\mathcal{P}} |g(\mathbf{X}, a_{\mathcal{P}})|$, tout en gardant la partie linéaire de manière exacte.

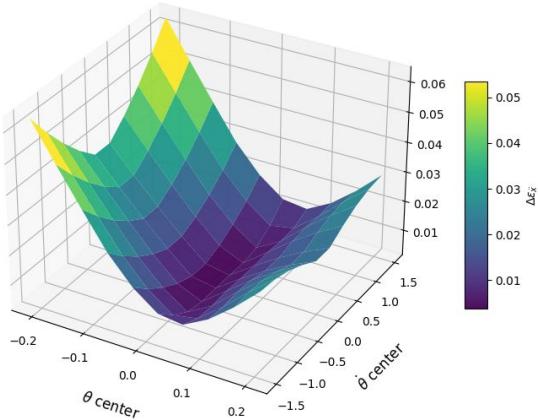
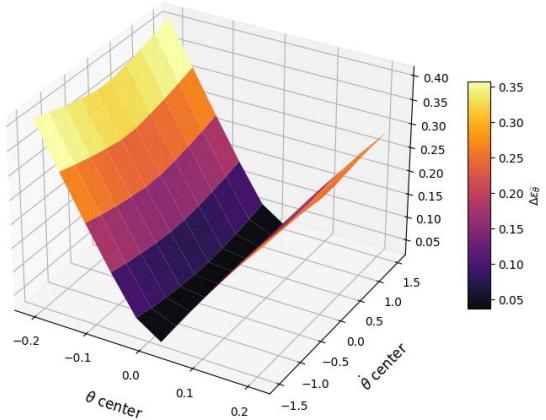
$$\epsilon_{\ddot{x}, \mathcal{P}}^{\max} = \max_{\mathcal{P}} |g(\mathbf{X}, a_{\mathcal{P}})_{\ddot{x}}|$$

Optimisation dans l'espace résiduel. Le programme linéaire s'exécute alors sur le résidu g uniquement, et \mathcal{P}_{ext} est translatées par $c_{\text{plong}}(a_{\mathcal{P}})$ pour réintégrer la partie affine.

Les vecteurs d'approximation α_i sont pre-multipliés par B de façon à isoler la partie non-linéaire :

$$\langle n \cdot \alpha \cdot B_{\text{plong}}, \mathbf{Z}_{\text{ext}} \rangle$$

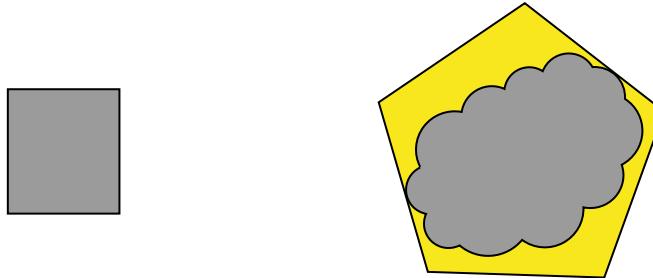
Et la variable ne vie plus dans \mathcal{P}_{ext} mais dans $\mathcal{P}_{\text{ext}} + c(a_{\mathcal{P}})$.

Variation of $\Delta\varepsilon_{\dot{x}}$ over local intervals(a) Ordre de grandeur de l'encadrement du résidu $\Delta\varepsilon_{\dot{x}}$ sur chacun des pavésVariation of $\Delta\varepsilon_{\dot{\theta}}$ over local intervals(b) Ordre de grandeur de l'encadrement du résidu $\Delta\varepsilon_{\dot{\theta}}$ sur chacun des pavésFIGURE 3.4. – Encadrement du résidu $\Delta\varepsilon$ sur les pavés

Differences entre les 2 encadrements. On observe en moyenne que l'encadrement du résidu, après soustraction de la composante linéaire (3.4), est deux fois plus étroit que l'encadrement direct du résidu (3.3).

Illustration géométrique

La Figure 3.5 synthétise visuellement la démarche : à gauche un polytope de référence, à droite l'approximation octogonale (contour jaune) englobant l'image non-linéaire (nuage gris).

FIGURE 3.5. – Illustration géométrique : Image réelle ($f_{\text{physique}}(\mathcal{P})$) de \mathcal{P} (carré gris) représenté par un nuage gris, dans son encadrement octogonal (octogone jaune)

Limites d'une borne uniforme sur le résidu

On a que g s'écrit :

$$g(\mathbf{X}, a_{\mathcal{P}}) = \begin{pmatrix} \frac{F_{\mathcal{P}} + m_p \sin \theta (\ell \dot{\theta}^2 + g \cos \theta)}{m_c + m_p \sin^2 \theta} - \left(\frac{m_p g}{m_c} \theta + \frac{(-1)^{a_{\mathcal{P}}} F_0}{m_c} \right) \\ \frac{-F_{\mathcal{P}} \cos \theta - m_p \ell \dot{\theta}^2 \cos \theta \sin \theta - (m_c + m_p) g \sin \theta}{\ell (m_c + m_p \sin^2 \theta)} - \left(\frac{(m_c + m_p) g}{m_c \ell} \theta + \frac{(-1)^{a_{\mathcal{P}}} F_0}{m_c \ell} \right) \end{pmatrix}.$$

On constate que g ne dépend que de θ et $\dot{\theta}$: $g(\mathbf{X}, a_{\mathcal{P}}) = g((\theta, \dot{\theta}), a_{\mathcal{P}})$

Une borne globale est sûre mais trop lâche dans certaine direction : pour certaines directions, ϵ est dicté par des sommets éloignés du domaine critique, ce qui n'est pas suffisant pour l'encadrement ou le raffinement.

Découpage directionnel. Ce qui peut-être fait, est de partitionner \mathcal{P} suivant différents découpage de $(\theta, \dot{\theta})$; chaque sous-polytope $\{\mathcal{P}^k\}_k$ reçoit sa propre borne $\epsilon_{\mathcal{P}^k}$, beaucoup plus étroite. On constate que lorsque la maille est 10 fois plus petite dans chacun de ces deux directions, la taille de l'encadrement est divisée par 10 (3.3.2).

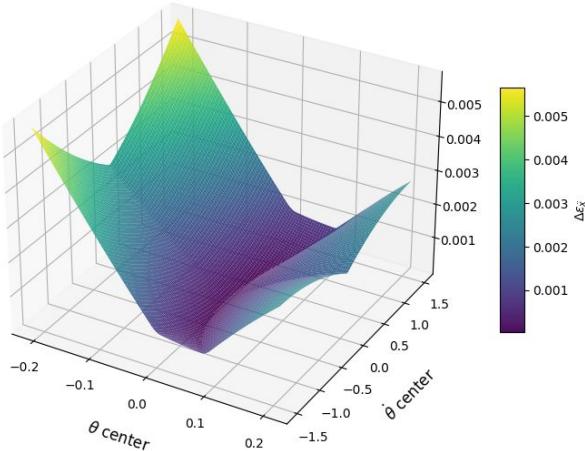
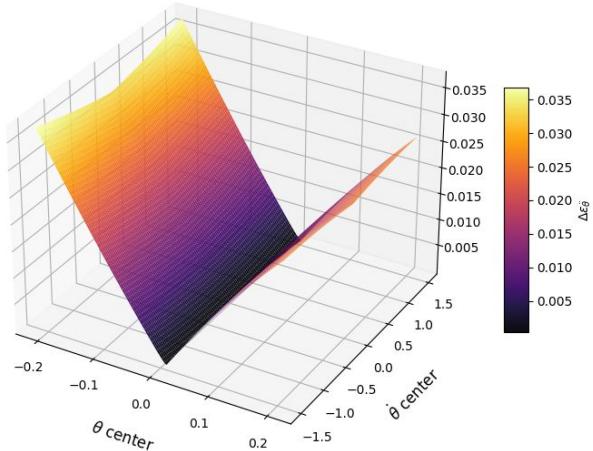
Variation of $\Delta\varepsilon_{\bar{x}}$ over local intervals(a) Ordre de grandeurde l'encadrement du résidu $\Delta\varepsilon_{\bar{x}}$ sur chacun des pavésVariation of $\Delta\varepsilon_{\bar{\theta}}$ over local intervals(b) Ordre de grandeurde l'encadrement du résidu $\Delta\varepsilon_{\bar{\theta}}$ sur chacun des pavés

FIGURE 3.6. – Effet du raffinement de la maille sur l'encadrement du résidu sur chacun des pavés : maillage dix fois plus fin

Proportionnalité à la maille h . Lorsqu'on affine le maillage dans chaque direction (de 10×10 découpes à 100×100 découpes), les erreurs sont globalement divisées par ≈ 10 . Cela suggère un comportement linéaire de l'erreur par rapport à la taille du pavé local, et ainsi confirme le fait de vouloir réduire la taille des pavés.

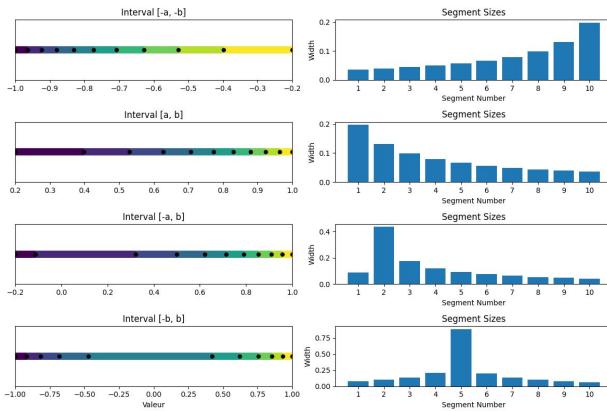
Découpage non linéaire pour la formation des pavés Dans un souci de justification de la pertinence de 3.3.5 et d'amélioration de la précision locale de l'approximation, notamment aux bords de l'espace des états où l'erreur est la plus élevée (voir figures 3.1a à 3.6b), j'ai exploré une stratégie de découpage non linéaire des intervalles, de façon à raffiner davantage les zones critiques.

Principe. Le principe repose sur une répartition non uniforme des points de découpe dans chaque dimension (figure 3.7a), induisant une densification des pavés aux extrémités des intervalles. Cela se traduit ensuite, en plusieurs dimensions, par une structuration asymétrique des polytopes (figure 3.7b) avec une concentration importante aux frontières.

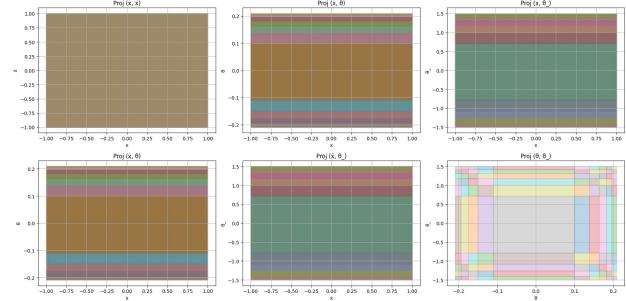
On peut constater que les figures 3.7c et 3.7d permettent de réduire d'environ 2,7 fois la taille de l'encadrement sur chacun des pavés, comparativement aux figures 3.4a et 3.4b, et ce pour un même nombre de découpages, mais avec une disposition différente.

Cependant, bien que prometteuse sur le papier, implémenter un maillage de chacun des polytopes à chaque étape s'est avérée peu concluante dans la pratique. Elle induit une explosion du nombre de pavés à manipuler, ce qui alourdit fortement le traitement géométrique (intersections, filtrages, etc.) et finit par ralentir l'algorithme sans y apporter un grand gain.

En conséquence, cette stratégie a été explorée durant les tests mais n'a pas été retenue comme un atout majeur dans la version finale de notre méthode ; et sert surtout à justifier la stratégie 3.3.5.



(a) Découpage non uniforme de segments 1D



(b) Découpage induit dans l'espace des états

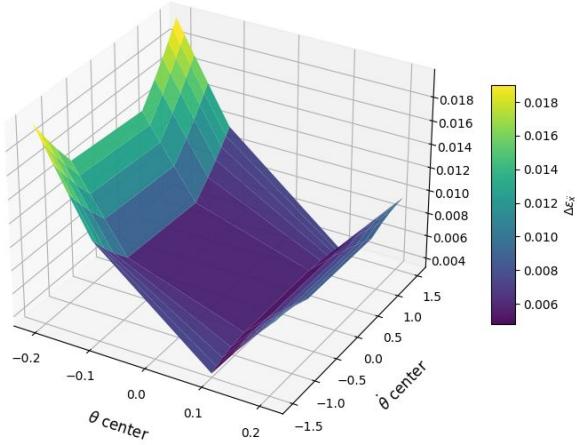
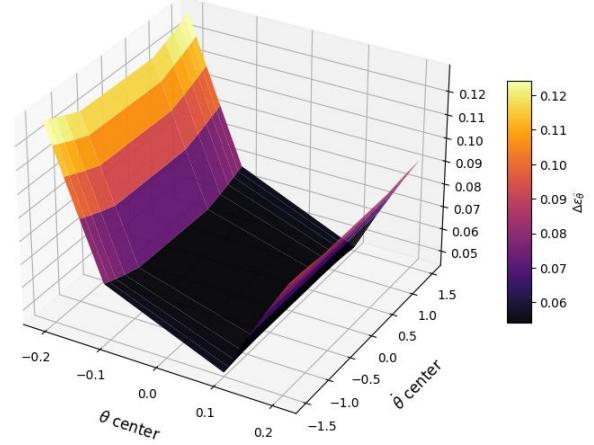
Variation of $\Delta\varepsilon_{\bar{x}}$ over local intervals(c) Ordre de grandeur de l'encadrement du résidu $\Delta\varepsilon_{\bar{x}}$ sur chacun des pavésVariation of $\Delta\varepsilon_{\bar{\theta}}$ over local intervals(d) Ordre de grandeur de l'encadrement du résidu $\Delta\varepsilon_{\bar{\theta}}$ sur chacun des pavés

FIGURE 3.7. – Approche par découpage non linéaire pour le raffinement adaptatif de l'espace d'états

3.3.3. Antécédents approximés (cas non linéaire)

Lorsqu'un point $y \in \mathcal{O}(\mathcal{I}(\mathcal{P}))$ est considéré comme dangereux et que l'on ne peut pas l'enlever, c'est qu'il est issu d'une trop grosse approximation d'une étape antérieure, il convient donc d'identifier les états de l'étape précédente susceptibles d'engendrer ce point *afin d'affiner le polytope de l'étape précédente*.

On a pu voir que dans le cas linéaire, l'opération d'antécédent s'obtient simplement par la décomposition du projeté du point sur les $\mathcal{S}(\mathcal{I}(\mathcal{P}))$, afin d'obtenir l'antécédent donné par le point avec les mêmes coefficients que cette décomposition mais avec $\mathcal{S}(P)$; la situation est plus délicate dans la dynamique non linéaire, à cause de la non-injectivité de f_{physique} .

Principe. On se place dans les hypothèses données par 3.3.1. Étant donné un point critique $y \in \mathbb{R}^4$, on cherche un ensemble $\mathcal{B} \subset \mathcal{P}$ tel que

$$f_{|\mathcal{P}}^{-1}(y) \subset \mathcal{B} \subset \mathcal{P}$$

Pour obtenir un tel ensemble, on procède comme tel :

1. **Sur-approximation de y** On encadre y à ϵ -près dans plusieurs directions : $\mathcal{P}_{\mathbf{Y}}^{\epsilon}$
2. **Problèmes linéaires** Le problème à résoudre est très proche de celui du calcul de \mathcal{OI} , cependant on optimise dans l'espace source :
 - a) On reprend les mêmes B_{plong} , $c_{\text{plong}}(a_{\mathcal{P}})$, \mathcal{P}_{ext} , $\mathcal{P}_{\text{plong}} = \mathcal{P}_{\text{ext}} + c_{\text{plong}}(a_{\mathcal{P}})$
 - b) On plonge la direction à optimiser dans \mathbb{R}^6 :

$$n_{\text{plong}} = \binom{n}{0}$$

- c) On ajoute la contrainte à $\mathcal{P}_{\text{plong}}$ pour obtenir $\mathcal{P}_{\text{constraint}} : \alpha \cdot B_{\text{plong}} \cdot \mathbf{X}_{\text{plong}}$ doit appartenir à $\mathcal{P}_{\mathbf{Y}}^\epsilon$
- d) On optimise

$$\langle n_{\text{plong}}, \mathbf{Z}_{\text{constraint}} \rangle$$

En prenant le \max_n et la direction n on construit un polytope qui encadre les antécédents de y
On a ainsi un encadrement grossier de l'ensemble des antécédents de y présent dans \mathcal{P}



FIGURE 3.8. – Recherche d'un antécédent : le point critique rouge dans l'image (polygone jaune) est trop proche de l'image réelle (nuage gris) et ne peut être plus optimisé (le problème vient donc d'avant), on encadre donc tous les antécédents (rectangle rouge) dans le polytope précédent (carré gris) dans l'espace source.

Cette méthode, conservatrice, permet d'exclure efficacement les régions initiales responsables d'états dangereux sans nécessiter d'approximation de l'inversion de f_{physique} . Cependant, cet encadrement peut être très grossier ; ainsi, des perspectives d'amélioration incluent l'utilisation d'un plus grand nombre de directions pour la sur-approximation.

3.3.4. Affinement itératif des octogones

Soit un polytope \mathcal{P} . Une fois l'ensemble $\mathcal{O}(\mathcal{I}(\mathcal{P}))$ construit, il peut encore contenir des points dangereux pour l'horizon de vérification ($n_{\text{max}} = 500$ pas). L'algorithme de raffinement de l'octogone vise à exclure ces états qui ne devraient pas être présent.

Étape 1 : Détection des sommets critiques On évalue la sûreté de chaque sommet extrémal $v \in \mathcal{S}(\mathcal{O}(\mathcal{I}(\mathcal{P})))$ par la fonction $\text{is_safe}(v, \pi, n_{\text{max}} - d) \in \{\text{safe}, \text{unsafe}\}$, qui simule la trajectoire de v avec la politique π sur $n_{\text{max}} - d$. Les sommets *unsafe* forment l'ensemble $\mathcal{U} \subset \mathcal{S}(\mathcal{O}(\mathcal{I}(\mathcal{P})))$.

Étape 2 : choix d'une direction de raffinement Pour chaque $u \in \mathcal{U}$, deux stratégies sont testées :

1. **Somme des normales actives.** Soit $\{A_i x \leq b_i\}$ la \mathcal{H} -représentation de l'octogone courant $\mathcal{O}(\mathcal{I}(\mathcal{P}))$, $\mathcal{K}(u) = \{i \mid A_i u = b_i\}$ l'ensemble des faces actives en u . On pose

$$\mathbf{n} = \sum_{i \in \mathcal{K}(u)} A_i$$

que l'on normalise pour obtenir une direction de coupe. (Se généralise au cas où la cible est un polytope)

2. **Direction image prédecesseur–cible.**

- a) On calcule un *antécédent* z^* :

- **Cas 1 : la cible est un point** $y_{\text{target}} \in \mathbb{R}^4$
 - On cherche un état $z \in \mathbb{R}^6$ tel que son image $\alpha B z$ soit au plus proche de y_{target} .
 - Le problème d'optimisation s'écrit :

$$\begin{aligned} \min_{z \in \mathbb{R}^6} \quad & \| \alpha B_{\text{plong}} z - y_{\text{target}} \|_2^2 \\ \text{sous la contrainte} \quad & z \in P_{\text{plong}} \end{aligned}$$

- **Cas 2 : la cible est un polytope** $\mathcal{T} = \{y \in \mathbb{R}^4 \mid A_{\mathcal{T}} y \leq b_{\mathcal{T}}\}$
 - On introduit une variable $y \in \mathbb{R}^4$ appartenant au polytope cible.

— Le problème d'optimisation devient :

$$\begin{aligned} \min_{z \in \mathbb{R}^6, y \in \mathbb{R}^4} & \| \alpha B_{\text{plong}} z - y \|_2^2 \\ \text{sous les contraintes } & z \in P_{\text{plong}} \\ & A_{\mathcal{T}} y \leq b_{\mathcal{T}} \end{aligned}$$

- b) Puis on prend $x^* = z_{|\mathbb{R}^4}^*$ $\mathbf{n} = \frac{u - f(x^*, a)}{\|u - f(x^*, a)\|}$. Cette direction peut-être plus robuste dans les zones « plates ».

Étape 3 : optimisation linéaire et coupe Soit \mathbf{n} la direction retenue. On optimise comme dans 3.3.2 où on prend la valeur du max. Si la distance obtenue entre u et la coupe reste $< \text{margin}$, on passe à la *technique de raffinement avec contrôle du } \varepsilon* (3.3.5), sinon on a réussi à l'éliminer et l'octogone est raffiné avec cette nouvelle équation.

Étape 4 : Boucle Une fois tous les points problématiques retirés, on recommence (car les découpages peuvent faire apparaître de nouveaux sommets), et ce jusqu'à ce qu'il n'y ait plus de sommets problématiques.

Pseudo-code condensé

```

vertices = extrema(oct)
unsafe    = [v for v in vertices if not is_safe(v, horizon)]

while unsafe and not going_up:
    for u in reverse(unsafe):
        n = choose_direction(u)           // (1) ou (2)
        b = compute(n, oct)              // calculation terminal b
        if cut_succeeds(n, u):          // summit eliminated
            octo.add(n)                // inequality added
        else:
            if not refine_via_predecessor(u, n, oct):
                going_up = true         // backtrack
                break
    unsafe = [v for v in extrema(oct) if not is_safe(v, horizon)]

```

Listing 3.1 – Raffinement des sommets non sûrs

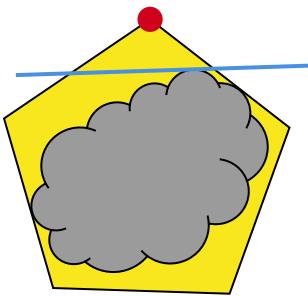


FIGURE 3.9. – Affinement : le sommet rouge est jugé non sûr ; une nouvelle coupe (trait bleu) est choisie pour resserrer l'octogone jaune et exclure le point (le nuage gris correspond à l'image réelle du polytope précédent).

3.3.5. Affinement avec contrôle fin du ϵ : problème de la borne trop lâche

Dans certains cas, un antécédent x^* (qui a l'une des images la plus proche de $u \in \mathcal{U}$), fournit une image qui est suffisamment éloignée du point problématique u pour fournir une bonne direction $\mathbf{n} = u - f_{\text{physique}}(x^*, a)$, mais la borne ε de la dérivée (qui est calculée sur l'ensemble du polytope \mathcal{P}) maintient les bords de l'octogone trop ample dans cette direction. Le correctif «raffinement avec contrôle du ε » procède ainsi :

1. **Demi-espace image.** Posons $b^{\text{img}} = \langle \mathbf{n}, f_{\text{physique}}(x^*, a) \rangle$. On restreint l'image courante à $O^{\text{pb}} = \{x \in \mathcal{O}(\mathcal{I}(\mathcal{P})) \mid \langle \mathbf{n}, x \rangle \geq b^{\text{img}}\} \neq \emptyset$ (car $x^* \in O^{\text{pb}}$).

2. **Boîte d'antécédents.** On procède comme dans 3.3.3, on calcule une boîte englobante \mathcal{Q} des points qui fournissent une image dans O^{pb}
3. **Nouvelle borne.** On évalue $\epsilon_{\mathcal{Q}}$ puis on optimise comme dans 3.3.2 où on prend la valeur du max $b^{\max \text{ opti}}$. Ensuite on pose $b^r = \max(b^{\max \text{ opti}}, b^{\text{img}})$. Ceci nous fournit un nouveau plan de coupe, puis on se ramène à la suite de l'algorithme : si la distance obtenue entre u et la coupe reste $< \text{margin}$, on backtrack, sinon on a réussi à l'éliminer et l'octogone est raffiné avec cette nouvelle équation.

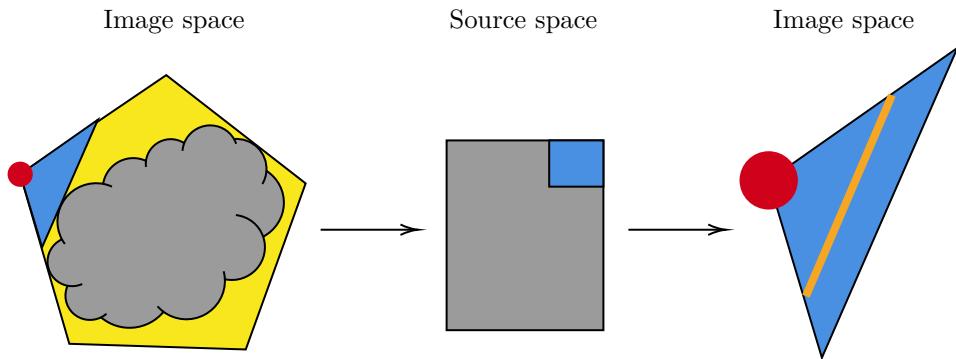


FIGURE 3.10. – Recalcule du ε et de la borne sur le polytope qui englobe l'ensemble des points qui donnent au moins une image dans la zone à raffiner

Extrait de code (simplifié)

```
# demi-espace image
img_pred = img(predecessor)
b_img = dot(n, img_pred)
cut_img = oct + {n, b_img}

# boîte d'antécédents
oct_pred = overapprox_predecessors(cut_img)

# nouvelle borne et plACEMENT dans R6
nouveau_eps = compute_eps(oct_pred)
oct_pred_plong = compute_plong(oct_pred, nouveau_eps)

#Calcul des nouveaux b
b_pred = max{oct_pred_plong}(n*alpha*B_plog, z_plong)
b_cut = max(b_img, b_pred)

If dist({n, b_cut}, probleme) < margin:
    bactrack(predecessor)
Else:
    oct.add({n, b_cut})
```

Listing 3.2 – Raffinement de l'abstraction avec contrôle de ε

3.4. Phase 3 – Suivi de l'ensemble des chemins

3.4.1. Nouveaux outils théoriques

Le suivi d'un chemin unique ne suffit pas à garantir la sûreté globale. L'intersection moyenne entre P_n et la partition induite par la politique contient *deux* polytopes, si bien que le nombre de feuilles croît en 2^n . Afin de maîtriser cette explosion combinatoire, différentes techniques ont été mises en place :

Clusterisation Regrouper les polytopes voisins dans l'espace des phases. Les expériences (Section 3.4.4 et 4.5) montrent qu'après 100 pas, l'univers se structure autour de 7–8 clusters stables à chaque étape.

Principe de récursion : Si un polytope P_n contient des ensembles issus de sous-polytopes visités à l'étape $n - i$, alors les descendants de ces ensembles n'ont pas besoin d'être suivis.

Il s'est avéré que je n'ai pas réussi à mettre en place des techniques performantes pour éliminer efficacement ces ensembles, qui ne sont pas nécessairement connexes ni convexes. Ainsi, je me suis contenté d'arrêter de suivre un polytope P_n s'il tombait dans un polytope déjà obtenu à une étape i (pour $i \in \{1, \dots, n-1\}$).

Démonstration du principe de récurrence On définit la partie « encore jamais vue » de P_k , que nous appelons son *essence* (E_k) :

$$E_0 := P_0 \quad D_k := \bigsqcup_{0 \leq i < k} E_i \quad E_k := P_k \setminus D_k \quad (k \geq 1)$$

On pose pour chaque entier $k \in \mathbb{N}$:

$$\mathcal{L}_k : \mathcal{B}(E_k) \longrightarrow \mathcal{B}(P_{k+1})$$

où notamment $\mathcal{L}_k(E_k) = P_{k+1}$

On construit \mathcal{L} (notre algorithme pris dans son ensemble) avec les $\{\mathcal{L}_k\}_k$, en posant :

$$\mathcal{L} : \bigsqcup_k \mathcal{B}(E_k) \longrightarrow \mathcal{B}(\mathbb{R}^4) \quad \mathcal{L}|_{E_k} = L_k$$

(cela revient à dire qu'à chaque étape k on applique \mathcal{L}_k)

On obtient (par récurrence) :

1. $\mathcal{L}(E_k) = \mathcal{L}_k(E_k) = P_{k+1}$
2. $\mathcal{L}(D_k) = \mathcal{L}(\bigsqcup_{0 \leq i < k} E_i) = \bigcup_{0 \leq i < k} \mathcal{L}(E_i) = \bigcup_{1 \leq i < k+1} P_i$
3. $\mathcal{L}(P_k) = \mathcal{L}(E_k \sqcup D_k) = \bigcup_{1 \leq i \leq k+1} P_i$

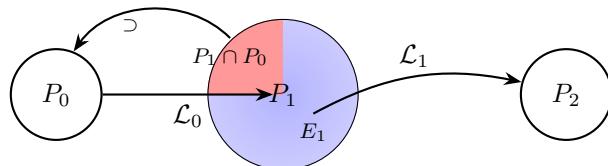
On obtient donc :

$$\mathcal{L} : \bigsqcup_k \mathcal{B}(E_k) \longrightarrow \bigcup_k \mathcal{B}(P_k) \tag{1}$$

Hypothèse globale (HG). Pour tout k , l'ensemble P_k est entièrement vérifié à l'étape k .

Conclusion. Sous cette hypothèse, la construction par récurrence de l'application \mathcal{L} garantit que chaque point de P_0 est certifié en fonction du nombre d'itération de \mathcal{L} (car selon 1, \mathcal{L} est à valeur dans $\bigcup_k \mathcal{B}(P_k)$, et tant que **HG** tient pour l'étape k).

En effet, même si à chaque étape k , \mathcal{L}_k n'agit que sur la partie encore inexplorée E_k et qu'on ignore les éléments déjà vus D_k , l'algorithme global agit sur l'ensemble, et donc tout point de P_0 est inclus dans une trajectoire suivie par l'algorithme.



3.4.2. Clusterisation des polytopes

Explosion combinatoire 2^n À chaque pas $n \rightarrow n+1$, chacun des polytopes suivis $\{\mathcal{P}_n^k\}_k$ coupe en moyenne deux régions distinctes de la partition induite par la politique π . Autrement dit, si l'on note

$$N_n = \#\{\text{polytopes effectivement suivis au pas } n\}$$

on a empiriquement

$$\mathbb{E}[N_{n+1} | N_n] \simeq 2N_n \implies \mathbb{E}[N_n] \approx 2^n N_0$$

Cette croissance exponentielle motive les stratégies de *clusterisation* et de *récurrence* décrites en 3.4.1, sans lesquelles la vérification dans le cas global serait impossible au-delà de quelques dizaines de pas.

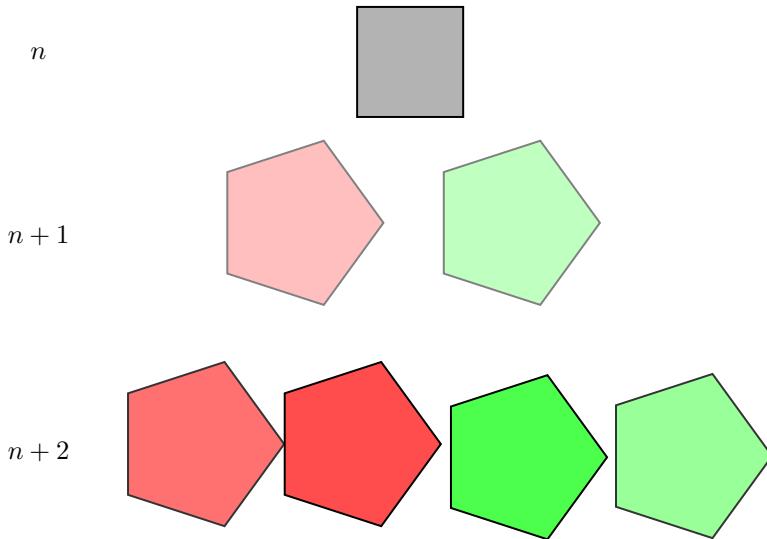


FIGURE 3.11. – Augmentation en moyenne par 2 : en l'absence de réduction ou de fusion, le nombre de polytopes suivis croît en $\mathcal{O}(2^n)$.

Clusterisation Pour maîtriser la croissance en 2^n décrite ci-dessus, nous regroupons (*clusterisons*) au pas n , après avoir fait le calcul des sur-approximations octogonales, les polytopes qui :

1. appartiennent à la *même* branche de la politique (même action $a \in \{0, 1\}$ au pas suivant $(n + 1)$) ;
2. sont voisins au sens d'une distance géométrique d_H (ou Hausdorff), via une clusterisation hiérarchicale.

Soit $\mathcal{C} = \{P_k^{(1)}, \dots, P_k^{(m)}\}$ un tel cluster. Comme chaque $P_k^{(i)}$ partage l'action $a_{\mathcal{C}}$, $\mathcal{O}(\mathcal{I}(\mathcal{C}))$ s'obtient en prenant le max dans les différentes directions de $\{\mathcal{O}(\mathcal{I}(P_k^{(1)})), \dots, \mathcal{O}(\mathcal{I}(P_k^{(m)}))\}$:

$$\mathcal{O}(\mathcal{I}(\mathcal{C})) = \left\{ x' \in \mathbb{R}^4 \mid \langle v, x' \rangle \leq \max_{i=1, \dots, m} M_v(P^{(i)}), \forall v \in \mathcal{V} \right\}$$

où \mathcal{V} est l'ensemble des vecteurs directions utilisés pour construire l'octogone et $M_v(P) = \max_{x \in \mathcal{O}(\mathcal{I}(P))} \langle v, x \rangle$ (ainsi, la borne est assez précise car chacun des $\mathcal{O}(\mathcal{I}(P))$ sont évalués *avec leur ε propre*). Autrement dit, on agrège les bornes supérieures *par prise de max*, ce qui évite de ré-optimiser sur l'enveloppe convexe de l'union $\bigcup_i P^{(i)}$ (ce qui augmenterait le ϵ en chaque point) tout en conservant la sûreté.

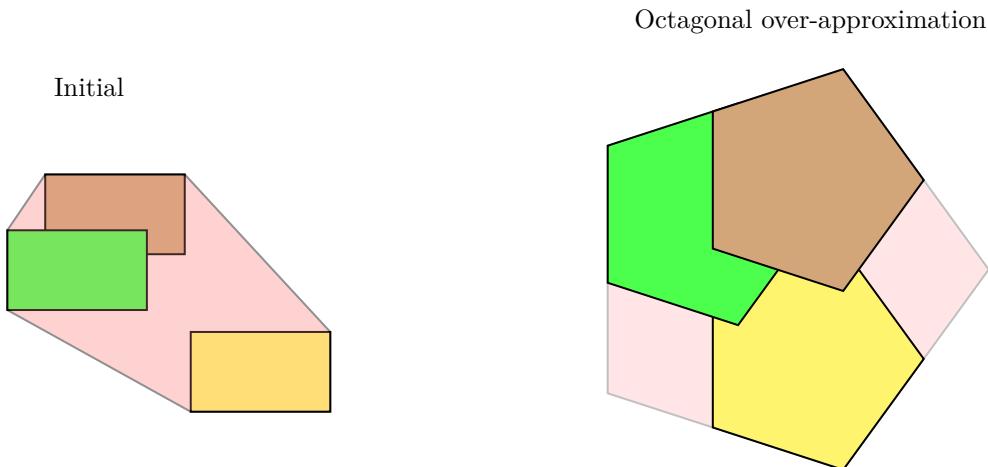


FIGURE 3.12. – Clusterisation : les polytopes partageant l'action et proches géométriquement sont fusionnés avant la phase d'over-approximation octogonale.

En moyenne on a 7 ou 8 clusters, ainsi cette opération abaisse drastiquement le nombre de polytopes suivis.
Algorithme de clusterisation hiérarchique

1. **Distance.** Calculer la matrice $D[i, j] = d_{\text{H}}(P^i, P^j)$, où :

$$d_{\text{H}}(P, Q) = \max \left(\max_{p \in S(P)} \min_{q \in S(Q)} \|p - q\|, \max_{q \in S(Q)} \min_{p \in S(P)} \|p - q\| \right)$$

2. **Linkage.** Appliquer un linkage hiérarchique

3. **Choix de k (Nombre de Clusters).** Balayer $k = 2, \dots, k_{\text{max}}$ et retenir

$$k_* = \arg \max_k S(k)$$

où $S(k)$ est le *silhouette score*.

4. **Clusters.** Répartir les polytopes selon les clusters correspondant au k choisi puis, pour chaque groupe, calculer l'enveloppe convexe.
5. **Recommencer.** Si une enveloppe convexe d'un cluster contient un point problématique on réapplique le principe de clusterisation uniquement sur ce cluster afin de réobtenir des clusters plus fins. (L'enveloppe convexe ne sert que pour cette partie de vérification)

Antécédents Le traitement des antécédents suit une logique en plusieurs étapes :

- Pour chaque point problématique, on cherche s'il admet un antécédent valide dans au moins un des polytopes sources (via la résolution de systèmes linéaires vu précédemment).
- Deux cas se présentent :
 - **Cas 1** : une solution est trouvée. Le point est donc considéré comme engendré par au moins un polytope précédent. On backtrack avec les nouveaux antécédents de chaque polytope.
 - **Cas 2** : aucune solution n'est trouvée. Cela signifie que le point ne correspond à aucun polytope antécédent. On suppose que ce point est une conséquence de la clusterisation (qui a regroupé plusieurs polytopes), et qu'il ne devrait pas exister à ce stade. Pour corriger cela :
 - on splitte le cluster correspondant,
 - puis on relance le procédé de raffinement individuellement sur les polytopes issus du split (Comme l'octogone image a déjà été calculé, cette relance est ciblée et ne nécessite pas de tout recalcul).

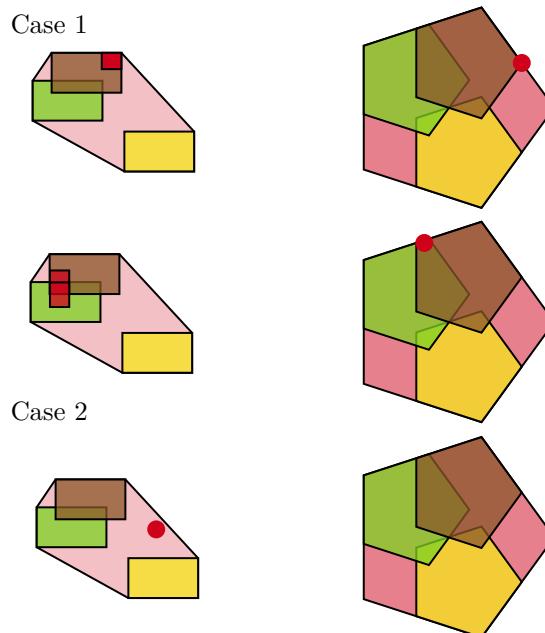


FIGURE 3.13. – Illustration des différents cas d'antécédents possibles

3.4.3. Suppression des polytopes déjà-vus (Récursion)

Dans un but de réduire le nombre de polytopes suivis, et même d'affiner les polytopes qui seront engendrés et suivis, il est très intéressant d'essayer d'enlever les ensembles qui ont été déjà vus car ceux-ci n'ont plus besoin

d'être vérifiés, grâce au principe de récurrence.

On observe cependant une certaine récurrence des configurations toutes les deux étapes. Ainsi, un premier algorithme simple a été implémenté : si un octogone produit à la date n est *entièrement* contenu dans un octogone déjà vérifié à la date $n - i$ avec $i \in \{0, \dots, 8\}$ et $n \equiv i \pmod{2}$, il n'est plus nécessaire de le suivre.

Algorithme

1. Conserver pour chaque parité (pair / impair) la liste des octogones vus sur les huit profondeurs précédentes.
2. Pour chaque nouveau polytope P_n :
 - a) si P_n est vide, on l'élimine ;
 - b) sinon, on teste $P_n \subseteq Q$ pour chaque Q ayant même parité et profondeur $\geq n - 8$.

Si l'inclusion est vraie, P_n est supprimé ; sinon, on l'ajoute à la structure et on continue son suivi.

Cette récurrence élimine en moyenne 25% des polytopes tout en conservant les principes de vérification.

3.4.4. Expérimentations du clustering et résultats sur la récursion

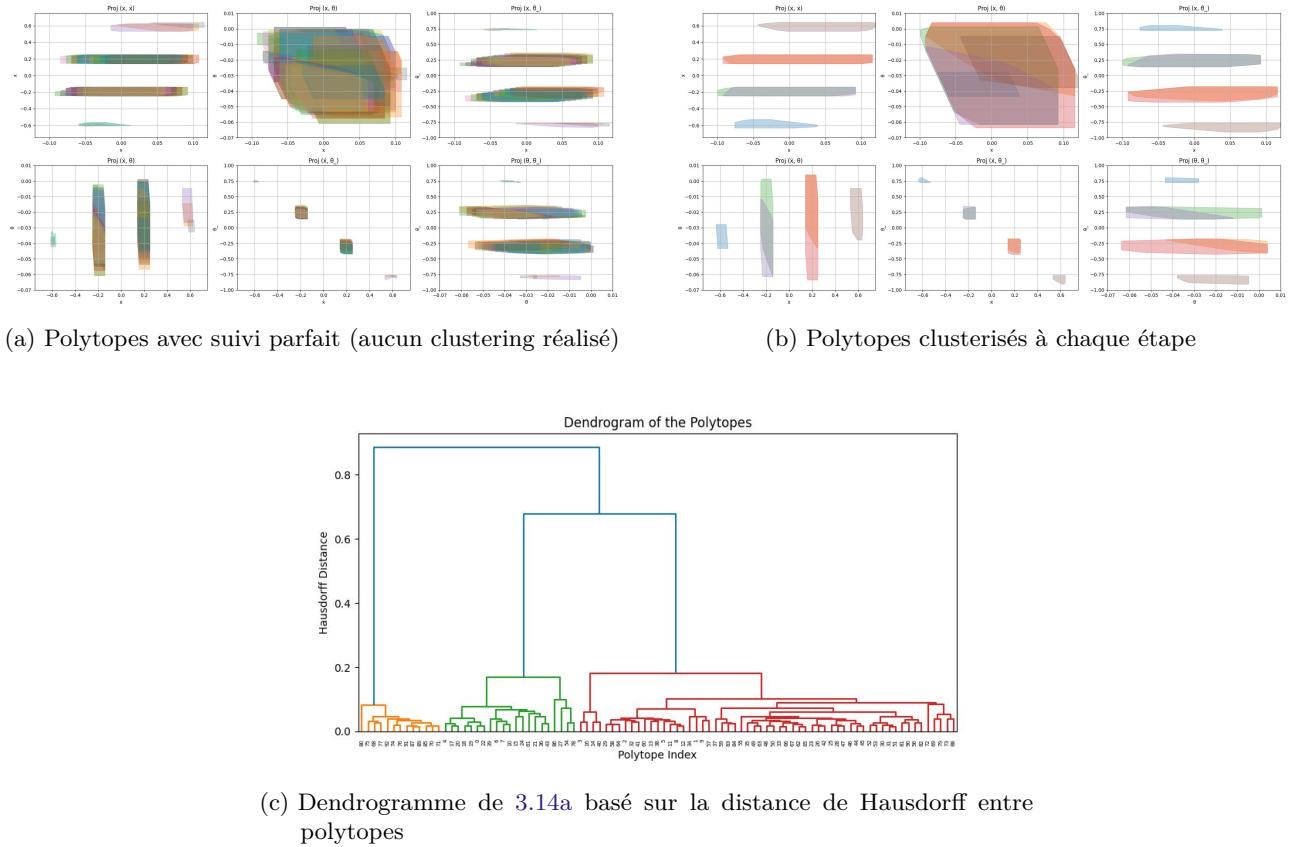


FIGURE 3.14. – Comparaison entre un suivi parfait et un suivi avec clusterisation, ainsi que le dendrogramme associé (correspondant à la profondeur 7)

Utilité de la clusterisation. Si l'on suivait parfaitement tous les chemins de propagation à chaque étape, on obtiendrait les polytopes de la figure 3.14a, correspondant à une description fine et complète de l'évolution de l'espace de décision. Cependant, cela devient rapidement intraçable à cause de la croissance exponentielle du nombre de branches. C'est pourquoi une approche par *clustering* est utilisée, comme illustré dans la figure 3.14b. Cette approche permet de limiter la complexité mais induit une perte de précision. En effet, à cause du problème de l' ε , les clusters ont tendance à d'avantage grossir.

Principe du dendrogramme. Le dendrogramme de la figure 3.14c illustre la clustérisation hiérarchique. Si l'on avait suivi l'approche complète, on aurait obtenu ce dendrogramme à l'étape 7, où l'on peut voir une séparation évidente en quatre groupes (niveau de coupe optimal). En pratique cependant, pour limiter les

effets liés à ε tout en évitant l'explosion combinatoire, on impose une coupe à une hauteur correspondant à $\min(\text{depth}, \text{nombre de polytopes})$. Cela permet de conserver un nombre de clusters raisonnable pour une profondeur donnée tout en gardant une bonne précision. C'est pourquoi, dans la figure 3.15, on observe un nombre élevé de polytopes. En effet, comme le clustering est effectué en regroupant des polytopes associés à même polytope d'action, plusieurs sous-groupes peuvent apparaître au sein d'un même regroupement (car les polytopes-actions peuvent différer). Par ailleurs, en imposant un nombre minimal de clusters — ici, environ 79 par sous-groupes — on se retrouve avec un nombre final de clusters suivis de l'ordre de $2 \times 79/6$, là où on aurait envie de suivre 6 clusters.

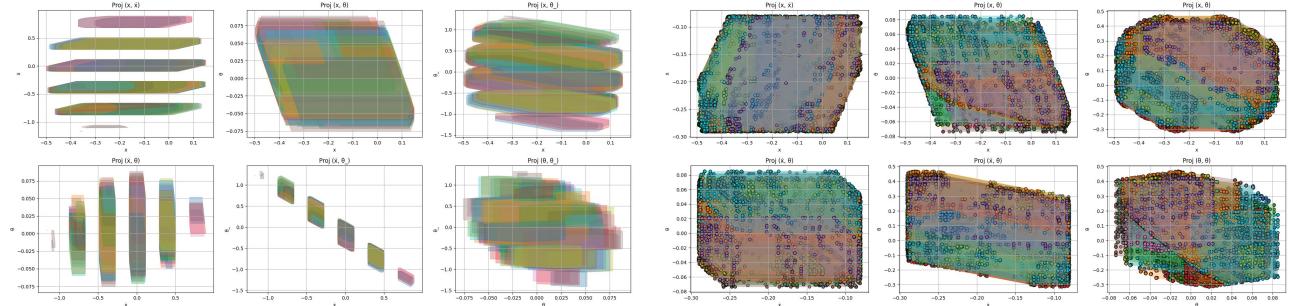
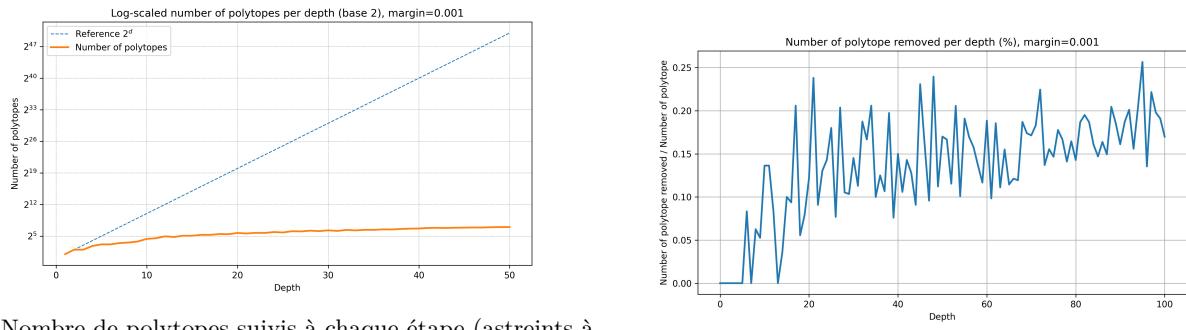


FIGURE 3.15. – Visualisation de l’ensemble des polytopes suivis à la profondeur 79 et un zoom sur les polytopes appartenant au cluster situé entre -0.02 et 0.02 selon \dot{x}

Valeurs choisies. Cette coupe contrôlée constitue également une condition pratique à l’application de la technique de récursion : en augmentant légèrement le nombre de clusters à chaque profondeur, les polytopes générés restent de taille contrôlée (ce qui compense l’augmentation induite par le paramètre ε) et permettent ainsi leur inclusion dans les polytopes des étapes précédentes.



(a) Nombre de polytopes suivis à chaque étape (astreints à une croissance minimale équivalente à celle de l’identité)

(b) Nombre de polytopes enlevés à chaque pas (%)

FIGURE 3.16. – Analyse de la dynamique du nombre de polytopes conservés et retirés à chaque profondeur

Résultats de la récursion. La figure 3.16 confirme que le nombre de polytopes croît en $O(\text{depth})$, ce qui reste maîtrisable. Par ailleurs, on observe qu'il est possible de supprimer en moyenne environ 15 % des polytopes, après raffinement via $\mathcal{I}(\mathcal{O}(\mathcal{P}))$ et intersection avec la politique du pas suivant.

3.4.5. Algorithme général de vérification CartPoleRL-Safe

L'algorithme Python complet étant volumineux, en voici une version pseudo-codée suffisamment détaillée :

```

# --- Initialisation -----
P0      = polytope_a_verifier
Depth   = 0
GoingUp = False          # back-tracking local
HistP   = {}              # historique des polytopes par profondeur
Remove  = {}              # cache pour la recurrence (+/-8 pas)
Problemes = {}            # cache pour stocker les problemes de chaque profondeur

# --- Boucle principale -----
while Depth < N_max:

    if GoingUp:
        # retour a la profondeur precedente
        Oct_List = HistP[Depth]
    else:
        # 1) Decoupage par la policy pi
        PiP_List = intersect_policy(PolyList)
        # 2) Image octogonale (dynamique non lineaire)
        Oct_List = octagon_image_parallel(PiP_List)
        HistP[Depth] = Oct_List

    GoingUp = False
    # 3) Affinage + detection d'etats dangereux (parallele)
    Oct_List, GoingUp, new_pb = parallel_raffinement(Oct_List, Problemes)
    if GoingUp:
        Depth -= 1;
        Problemes.update(Depth, new_pb)
        continue

    # 4) Suppression par recurrence temporelle
    Oct_List, Remove = manage_remove(Oct_List, Remove, Depth)

    # 5) Clusterisation
    PolyList = manage_merging(Oct_List)

    # 6) Passage a la profondeur suivante
    Depth += 1

# --- Sortie -----
if Depth == N_max:
    print("SAFE up to horizon")
else:
    print("UNCERTAIN / TIMEOUT / MEMORY")

```

Listing 3.3 – Boucle principale de propagation et de raffinement

Commentaires

- `parallel_raffinement`, `manage_remove` et `manage_merging` encapsulent respectivement : l'affinage des octogones et donc la vérification que l'octogone obtenu est dans les bornes de suretées (Section 3.3.4), la suppression par le principe de récurrence (Section 3.4.3) et la clusterisation (Section 3.4.2).
- Le drapeau `GoingUp` assure le suivi du retour en arrière : en cas d'échec d'affinage à la profondeur d , on remonte à $d - 1$ pour raffiner les parents.
- Tous les calculs lourds (images, sûreté, distances) sont exécutés en parallèle.
- Un système de mémoisation a été mis en place pour retrouver efficacement les différents états aux différentes profondeurs, et limiter le nombre de recalculs sans exploser la capacité de stockage.

4. Résultats

Les résultats présentés dans cette section concernent l'environnement CartPole-v1, déjà introduit précédemment. Afin d'évaluer la capacité de notre algorithme à vérifier les trajectoires du polytope initial et comprendre son fonctionnement, il y aura principalement 2 types de représentation des représentations de projection de polytope sur 6 plans 2D, pour comprendre les images et encadrement que réalise l'algorithme à chaque étape ; et des graphiques d'analyses des performances de notre algorithme.

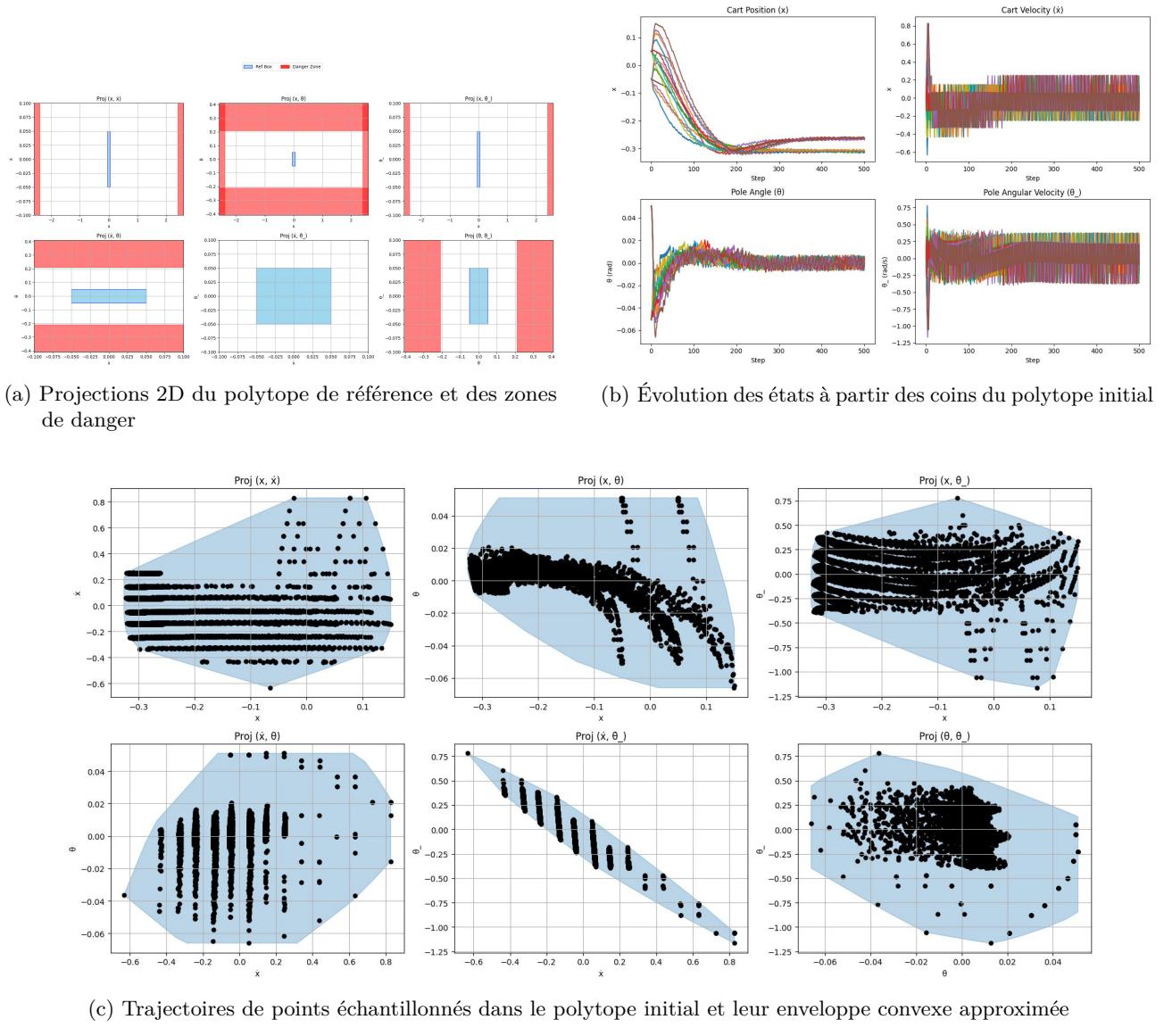


FIGURE 4.1. – Visualisations associées au polytope de référence

Tout d'abord, pour bien comprendre la situation initiale et dans quel espace vit notre polytope initial, voici figure 4.1a qui représente le **polytope de référence** (en bleu clair), soit l'hyperrectangle $[-0.05, 0.05]^4$, projeté deux à deux sur les différentes paires de dimensions $(x, \dot{x}, \theta, \dot{\theta})$; tandis que les zones en rouge matérialisent les **zones de danger**, c'est-à-dire les régions de l'espace d'état qui violent les contraintes de sûreté du système.

De plus, pour comprendre les trajectoires qu'il est nécessaire d'encadrer, la figure 4.1b illustre les trajectoires complètes simulées à partir des **coins du polytope de référence**. Pour chaque coin de l'hypercube initial, l'évolution du système est tracée sur 500 étapes. Ces courbes permettent ainsi de visualiser les dynamiques des

états limites, et de comprendre comment se comportent les états extrêmes étudiés. Ainsi, on peut notamment voir que les 100 premiers pas semblent être les plus volatiles puis ensuite, l'ensemble semble converger vers une zone d'équilibre.

Enfin, encore dans l'optique de comprendre dans quel espace vivent les trajectoires que l'on cherche à encadrer, la figure 4.1c montre les **trajectoires de points échantillonnés** dans l'intérieur du polytope de référence, projetées sur les différents plans 2D. Le fond bleu représente une *pseudo-enveloppe convexe* construite à partir des points visités au cours des trajectoires. Ainsi, on peut voir que cet ensemble est assez compacte, et semble être bien inclu dans la zone sûre (même une zone sûre plus étroite). De plus, cette figure de l'enveloppe convexe réapparaîtra dans les résultats des différentes expérimentations.

Attention toutefois à l'interprétation de ces graphiques : chaque figure représente une projection bidimensionnelle d'un espace à 4 dimensions. Il est donc possible qu'un point apparaisse dans une zone « sûre » en projection, alors qu'il viole une contrainte dans une dimension non affichée. L'analyse complète doit tenir compte de cette limitation.

4.1. Certification d'un chemin en physique non-linéaire

Ma première série d'expériences a consisté à certifier un chemin de 500 étapes dans un environnement de physique non-linéaire. Deux configurations ont été comparées : l'une suivant barycentre du polytope initial, et l'autre choisissant comme point initial le sommet maximisant ($x_1 + x_3$).

Cas 1 : Point central (barycentre)

Dans ce cas, l'algorithme a nécessité 904 itérations pour certifier une trajectoire de profondeur 500, soit environ 27 minutes sur un CPU 16 coeurs. Le polytope tend à occuper pleinement l'espace admissible dans les directions contraintes. On note des rétropropagations fréquentes (*backtracks*) et quelques plateaux dans la croissance de la profondeur. Le nombre de contraintes et le volume restent néanmoins stables, sans explosion.

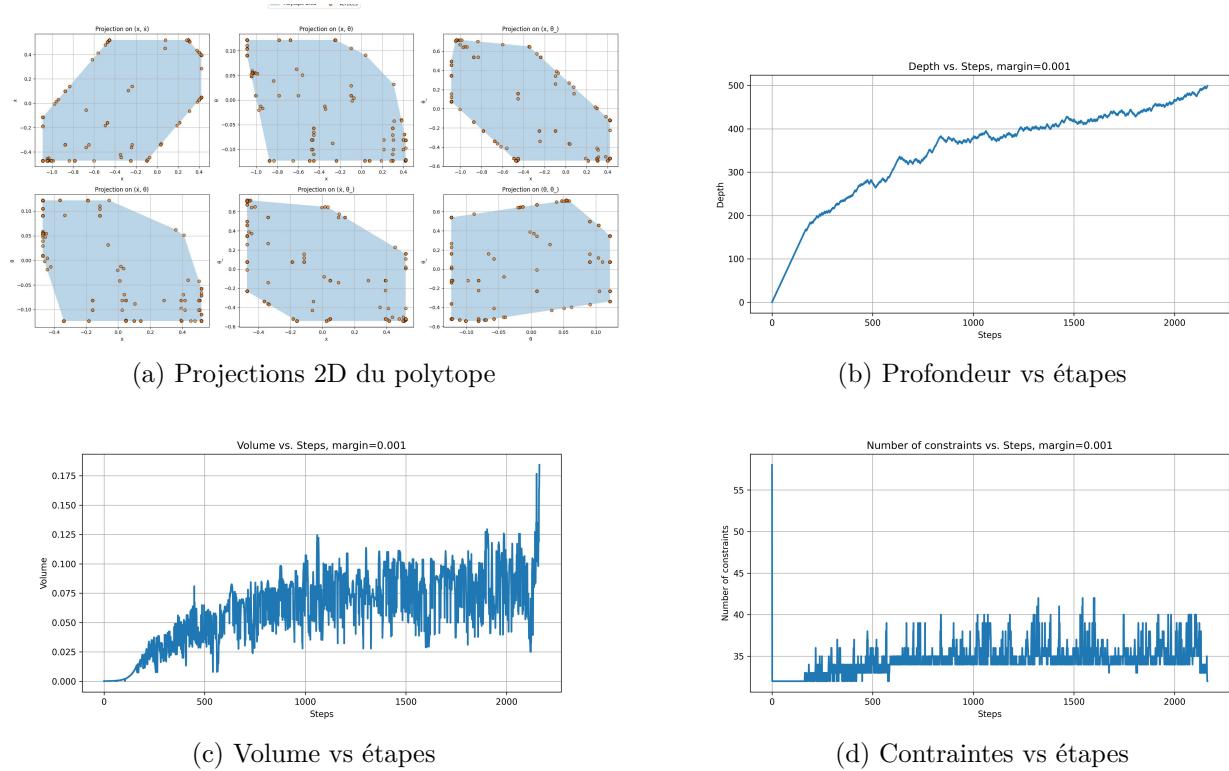


FIGURE 4.2. – Comportement de l'algorithme avec point central (marge = 0.001)

Cas 2 : Point ($x_1 + x_3$)_{max}

Dans ce second cas, le point de suivi est choisi de manière extrémale. Ce choix a conduit à une trajectoire sans aucun backtrack, ce qui traduit une meilleure stabilité locale du polytope. On observe une décroissance rapide

du volume en début de trajectoire, ce qui peut s'expliquer par le fait que les polytopes de décision sont plus petits dans les zones extrémiales.

Une fois que le volume est suffisamment faible, le polytope semble se stabiliser dans une configuration «cohérente» : les contraintes se stabilisent, la profondeur augmente linéairement, et l'algorithme converge rapidement sans rupture.

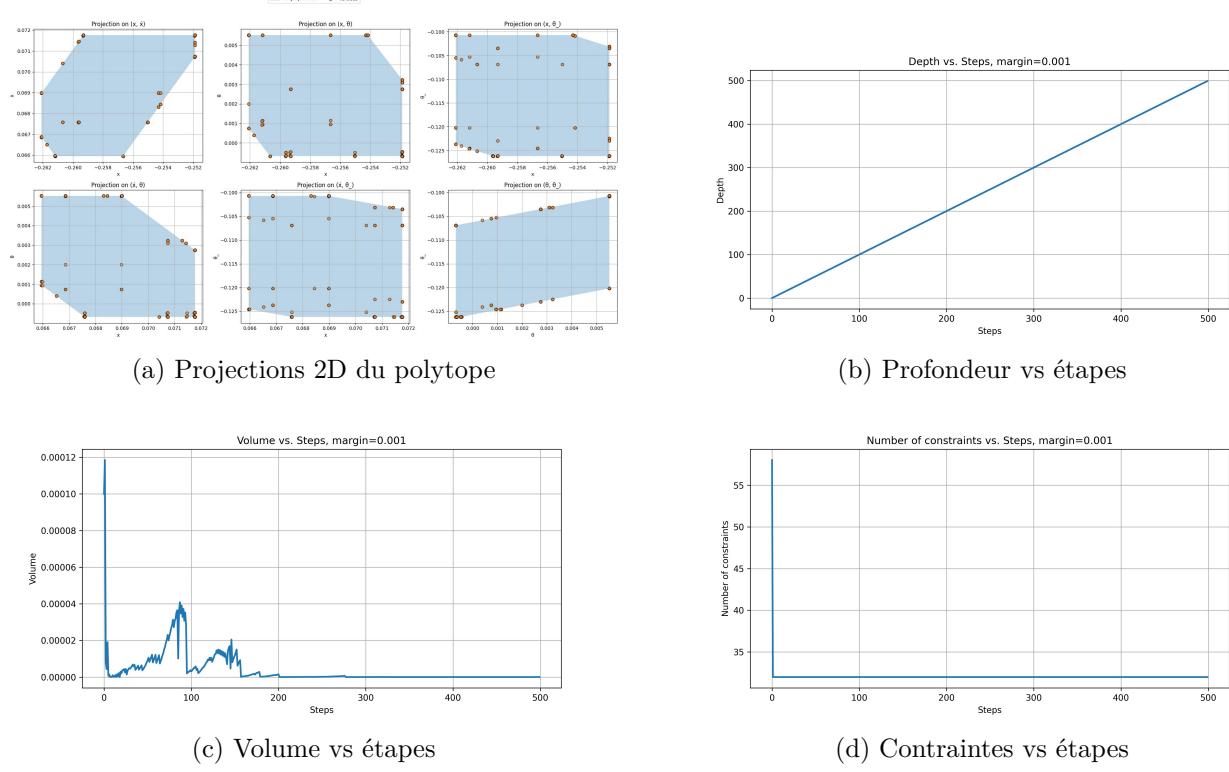


FIGURE 4.3. – Comportement de l'algorithme avec point $(x_1 + x_3)_{\max}$ (marge = 0.001)

4.2. Certification globale en physique non-linéaire

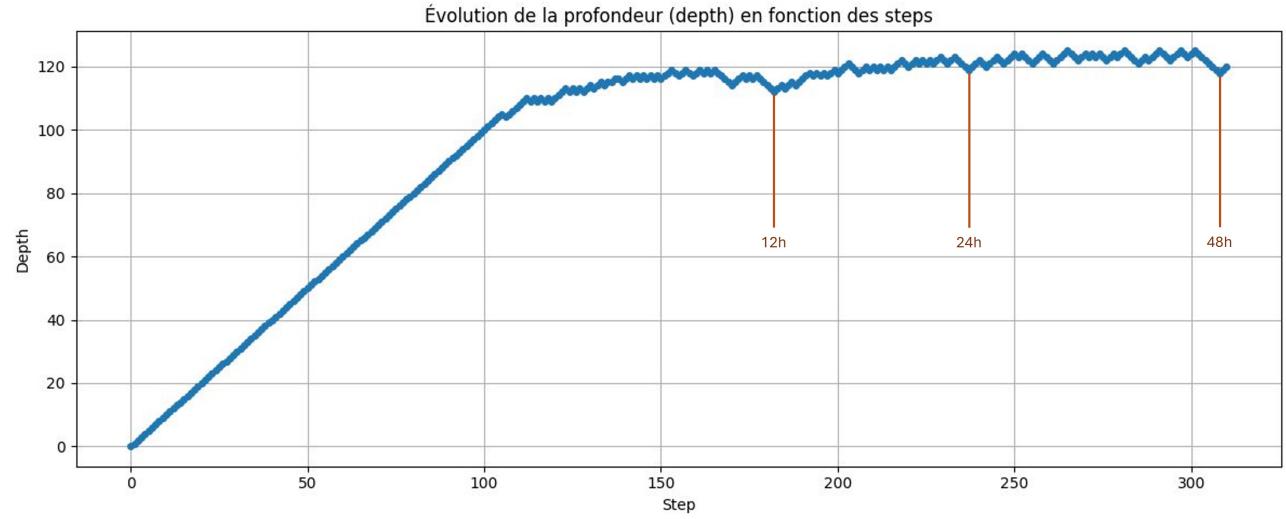


FIGURE 4.4. – Évolution de la profondeur en fonction des étapes jusqu'à la profondeur 125.

Le principal résultat obtenu est la certification globale jusqu'à la profondeur 125, atteinte après deux jours de calcul sur un CPU à 64 coeurs. La figure 4.4 montre qu'à partir de la profondeur 118, l'algorithme entre dans une phase de stagnation, la progression continue mais devient extrêmement lente (beaucoup de corrections via backtracking).

Cette observation renforce l'intérêt d'approfondir l'approche basée sur le calcul de polytopes intérieurs, en ne conservant que les résidus réellement actifs, afin de maîtriser la complexité croissante.

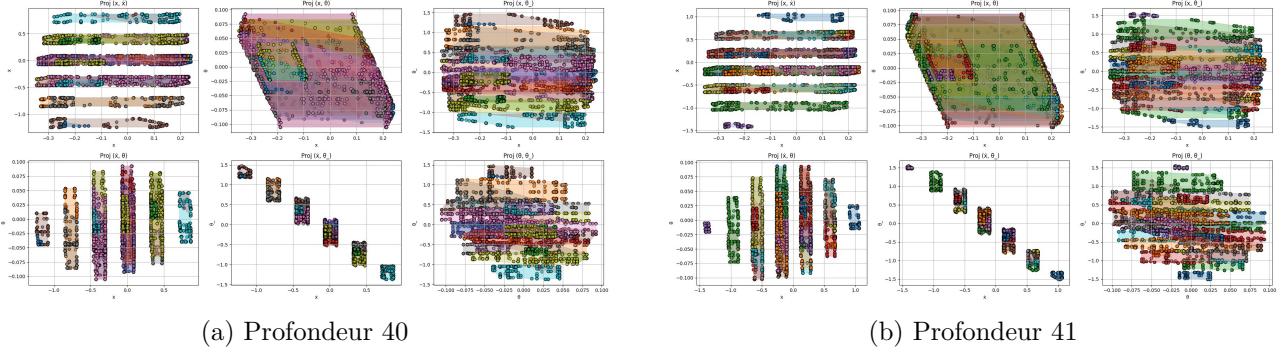


FIGURE 4.5. – Polytopes suivis respectivement aux profondeurs 40 (gauche) et 41 (droite).

La figure 4.5 illustre le comportement de l'algorithme à deux étapes consécutives. Ce dernier agit comme une marche aléatoire guidée, avec une propagation continue dans les dimensions x et θ , tandis que les dimensions \dot{x} et $\dot{\theta}$ évoluent de manière plus discrète, du moins sur les pas observés jusqu'à présent.

On remarque que le nombre de clusters reste relativement stable : environ 6 clusters pour les étapes paires, et 7 pour les étapes impaires. Cette alternance persiste à mesure que la profondeur augmente, bien que la croissance du nombre de clusters ralentisse très rapidement. Cela s'explique par le fait que l'algorithme atteint des configurations de plus en plus éloignées du polytope initial, souvent marginales ou peu pertinentes à traiter.

5. Discussion

5.1. Limites du travail

Les avancées obtenues — certification jusqu'à 125 pas dans le modèle non linéaire complet de *CartPole* — constituent un jalon significatif, mais elles s'accompagnent de limitations qu'il convient d'expliciter afin de guider les travaux futurs.

Précision numérique et cohérence des trajectoires. La gestion du paramètre ε entraîne une sur-approximation parfois trop lâche des volumes images, dégradant la performance de l'algorithme. Plus largement, l'efficacité des résultats dépend fortement des hyper-paramètres (tolérances numériques, marge, etc.), dont le réglage demeure empirique. Les erreurs d'arrondi amplifiées par les solveurs peuvent ainsi faire apparaître des polytopes « fantômes » ou masquer des violations réelles, altérant la fiabilité de l'algorithme.

Puissance de calcul et complexité exponentielle. L'exploration globale jusqu'à la profondeur 125 a mobilisé 64 coeurs pendant deux jours, soulignant la nécessité d'e continuer à optimiser le nombre de polytopes et de contraintes. Le parallélisme — indispensable — ne suffit pas lorsque les backtracks deviennent majoritaires : chaque retour en arrière déclenche un recalculation global coûteux. Une refonte de cette phase ou une stratégie « locale » de backtracking pourrait réduire drastiquement ce goulot d'étranglement.

Spécialisation du cadre expérimental. Le binôme *CartPole* + représentation polytopique facilite la visualisation et l'instrumentation, mais restreint la générativité. Les dynamiques physiques plus fortement non linéaires (drones 3D, robots manipulateurs) ou les espaces d'états de grande dimension nécessiteraient une adaptation en profondeur.

5.2. Perspectives

Au cours du stage, une pistes d'amélioration a été amorcée sans atteindre son plein potentiel et une autre piste envisagée. Elles sont détaillées ci-dessous, accompagnées de prolongements envisageables.

5.2.1. Polytopes intérieurs et récursion maîtrisée

Bilan : un prototype prometteur, mais encore coûteux. Réalisation d'un algorithme d'extraction d'un *polytope intérieur* strictement inclus dans l'union des polytopes d'un cluster (Figures 5.2–5.4). En soustrayant ce volume « déjà-vu » à l'ensemble à certifier, on réduit le nombre de résidus actifs et l'on limite la croissance des clusters, avec à la clé un contrôle plus fin du ε (cf 3.4.3). Les premiers essais en 2D confirment l'intérêt de cette approche ; toutefois, le passage en dimension 4 s'avère lourd : plusieurs minutes sont nécessaires pour un seul polytope, rendant l'intégration dans des runs longs irréaliste à ce stade.

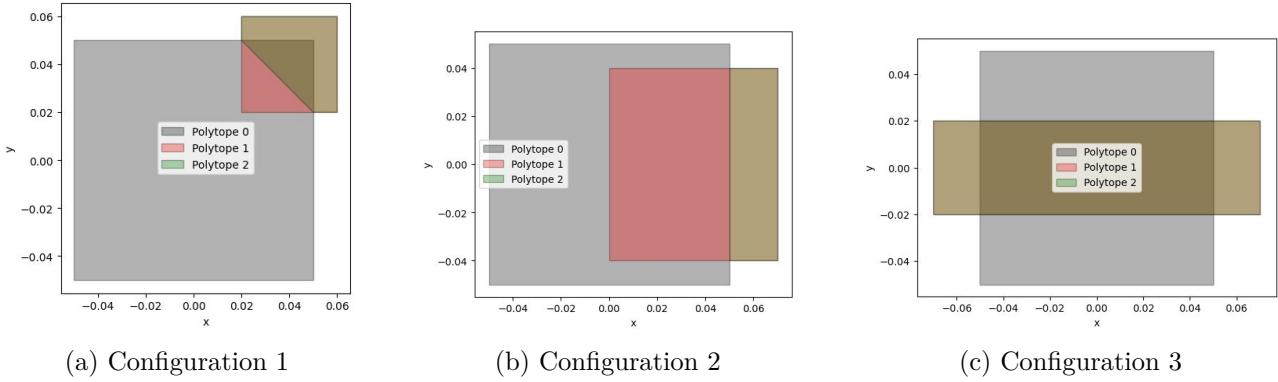


FIGURE 5.1. – Soustraction du polytope gris (intérieur) au polytope vert : la partie retirée apparaît en rouge, la partie conservée en doré.

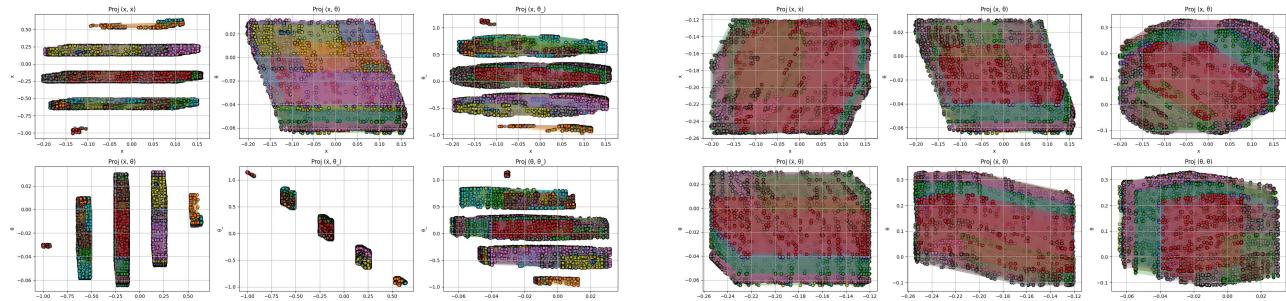


FIGURE 5.2. – Exemple d'ensemble : Ensemble des polytopes vus aux profondeurs $n = 8, n = 6, n = 4, n = 2$ pour $n = 30$ et zoom sur les polytopes intersectant $[-0.02, 0.02]$ selon \dot{x} .

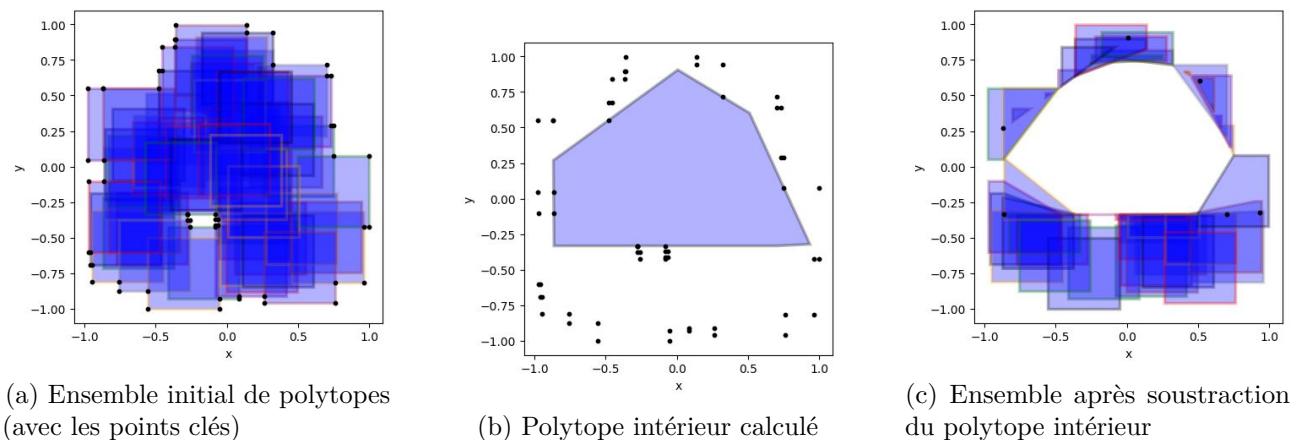


FIGURE 5.3. – Étapes d'identification et d'extraction du polytope convexe et connexe intérieur.

Méthode La méthode actuelle identifie les points clefs de l'ensemble et pousse l'enveloppe convexe vers l'intérieur (en adaptant le décalage pour chaque hyperplan de manière optimisée) en se basant sur un point de référence, choisie de manière idoine.

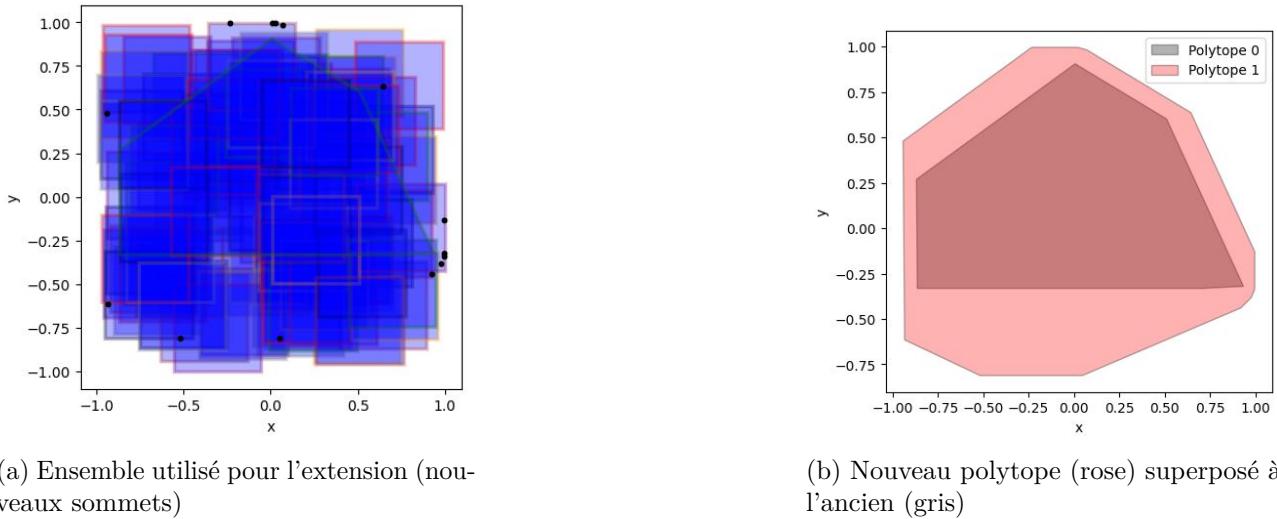


FIGURE 5.4. – Extension incrémentale du polytope intérieur : volume supplémentaire obtenu.

Méthode incrémentale Une méthode incrémentale a été implementée sur la méthode précédente afin d'étendre ce polytope à chaque étape, en garantissant les inclusions successives, nécessaire à notre algorithme.

Pistes d'exploration.

- **Optimisation algorithmique** : reformuler l'étape d'identification des points clés pour la rendre plus rapide.
- **Vérification des trous** : l'algorithme est quasiment sûr de ne pas avoir de trous à la première passe dans un clusters compacte, cependant afin de certifier la sûreté de manière formelle il est nécessaire de mettre en place une itération de vérification qui est couteuse.

5.2.2. Polytopes établis par la physique ou l'apprentissage

Une seconde piste consiste à repérer des ensembles de trajectoires qui sont quasi-stable dans l'espace des phases — autrement dit, des sortes d'«orbites» — et à s'en servir comme points d'appui pour la certification. L'idée est de capitaliser sur ces régions où le système se comporte de façon prévisible et répété pour simplifier les zones à vérifier grâce au principe de récursion.

1. **Observation directe des orbites.** Identifier ces trajectoires quasi répétitives à partir des simulations, puis les résumer sous forme de contraintes simples (polytopiques) qui peuvent être injectées dans la boucle de vérification.
2. **Apprentissage assisté.** Entraîner un petit modèle pour identifier les zones quasi-stable, avant de convertir cette information en contraintes linéaires utilisables par le vérificateur.

5.3. Apport personnel

1. **Utilisation de solveurs linéaires.** L'intégration d'un solveur linéaire dans l'algorithme de certification m'a permis de formuler les contraintes sous forme polytopique, puis de les résoudre efficacement via des programmes linéaires (PL). J'ai appris à reformuler un problème géométrique de façon algébrique et à manipuler les résultats du solveur pour mieux contrôler ce qui se passe dans l'algorithme.
2. **Géométrie dans l'espace.** Ce projet m'a amené à manipuler des objets en dimension 4, ce qui m'a conduit à utiliser des notions de géométrie dans l'espace, notamment les enveloppes convexes, les projections, la distance de Hausdorff, ainsi que du calcul vectoriel.
3. **Contrôle de fonctions dynamiques.** J'ai dû bien contrôler la fonction d'évolution pour garantir la certification des trajectoires. Cela m'a amené à encadrer de nombreux résidus afin de limiter les erreurs, et à surveiller attentivement les erreurs numériques et les tolérances des solveurs pour que l'algorithme reste fiable.

4. **Calcul sur des ensembles convexes.** J'ai appris à construire des approximations convexes (comme des octogones) pour représenter les images de polytopes, à gérer les notions de contraintes et d'hyperplans, et à utiliser ces outils pour simplifier les calculs tout en conservant des garanties de sûreté.
5. **Mise en place d'une clusterisation.** Pour limiter le nombre de polytopes, j'ai mis en place une méthode de regroupement en fonction de leur distance de Hausdorff. Cela m'a permis de mieux gérer la complexité sans trop perdre en précision. Toute cette partie m'a permis d'approfondir les notions de clusterisation hiérarchique et de les mettre concrètement en pratique.
6. **Certification d'un agent RL.** J'ai étudié un agent CartPole pour comprendre ses choix et les valider formellement. Cela m'a permis de mieux saisir comment se représentent les actions dans un agent d'apprentissage par renforcement, et de me confronter aux problématiques d'explicabilité et de certification de sûreté.

6. Conclusion

Au terme de ce stage, les résultats obtenus montrent qu'il est possible d'étendre significativement l'horizon de sûreté dans un environnement non linéaire en combinant des agents interprétables à des techniques de certification polytopiques. Le franchissement de la barre symbolique des 125 pas laisse entrevoir une généralisation vers le cap des 500 pas, sous réserve de lever les verrous algorithmiques identifiés. Les axes d'amélioration explorés — tels que l'accélération de la construction des polytopes intérieurs ou l'exploitation plus fine de la structure physique de l'environnement — ouvrent des perspectives de recherche encore largement prometteuses.

Sur le plan personnel, ce projet m'a permis de consolider plusieurs compétences clés : modélisation géométrique, formulation et résolution de programmes linéaires, ainsi que méthodologie de vérification formelle appliquée à l'apprentissage par renforcement. J'ai acquis une maîtrise concrète des solveurs numériques, approfondi ma gestion des erreurs et renforcé ma compréhension des mécanismes de certification des agents autonomes. Plus largement, cette expérience a nourri ma capacité à articuler performance algorithmique, interprétabilité et exigence de sûreté.

A. Annexes

Notations

- $\mathcal{P} \subset \mathbb{R}^4$: polytope convexe.
- $\mathcal{S}(\mathcal{P})$: ensemble des sommets de \mathcal{P} .
- $f_{\text{physique}}(\mathbf{X})$: fonction qui un état $\mathbf{X} \in \mathbb{R}^4$ associe son image dans \mathbb{R}^4 après action de la physique non-linéaire
- $\mathcal{I}(\mathcal{P})$: image de \mathcal{P} par la dynamique physique, i.e. $\mathcal{I}(\mathcal{P}) = f_{\text{physique}}(\mathcal{P})$.
- $\mathcal{O}(\mathcal{P})$: sur-approximation orthotopique (ou octogonale) de \mathcal{P} .
- $\mathcal{L}(\mathcal{P})$: image de \mathcal{P} par l'algorithme, i.e. $\mathcal{L}(\mathcal{P}_n) = \mathcal{P}_{n+1}$.
- $\mathcal{A}(\mathcal{P})$: ensemble des antécédents de \mathcal{P} par la dynamique physique, i.e. $\mathcal{A}(\mathcal{P}) = \{x \mid f_{\text{physique}}(x) \in \mathcal{P}\}$.

Bibliographie

- [1] Bastani, O., Pu, Y., & Solar-Lezama, A. (2018). *Verifiable Reinforcement Learning via Policy Extraction*. In Advances in Neural Information Processing Systems (NeurIPS). arXiv :1805.08328.
- [2] Pandia, S. P., Genest, B., Easwaran, A., & Suganthan, P. N. (2024). *Vanilla Gradient Descent for Oblique Decision Trees*. arXiv :2408.09135.
- [3] Marton, M., Genest, B., et al. (2025). *SYMPOL : Symbolic Policy Learning via Policy Gradient in Interpretable Tree Space*. In International Conference on Learning Representations (ICLR), à paraître.
- [4] Vasic, M., Petrovic, A., Wang, K., Nikolic, M., Singh, R., & Khurshid, S. (2022). *MoËT : Mixture of Expert Trees and its application to verifiable reinforcement learning*. arXiv :1906.06717.
- [5] Schilling, C., Lukina, A., Demirović, E., & Larsen, K. G. (2023). *Safety Verification of Decision Tree Policies in Continuous Time*. In Proceedings of the 37th NeurIPS Conference. URL : <https://openreview.net/forum?id=tEKBu5XOTw>.
- [6] Prajna, S., & Jadbabaie, A. (2004). *Safety verification of hybrid systems using barrier certificates*. In *Hybrid Systems : Computation and Control (HSCC)*, LNCS 2993, Springer, pp. 477–492.
- [7] Luo, Y., & Ma, T. (2021). *Learning Barrier Certificates : Towards Safe Reinforcement Learning with Zero Training-time Violations*. arXiv :2108.01846.
- [8] Mandal, U., et al. (2024). *Formally Verifying Deep Reinforcement Learning Controllers with Lyapunov Barrier Certificates*. In *Formal Methods in Computer-Aided Design (FMCAD)*.
- [9] Zhao, H., Zeng, X., Chen, T., & Liu, Z. (2020). *Synthesizing Barrier Certificates Using Neural Networks*. In *Proceedings of the 23rd ACM International Conference on Hybrid Systems : Computation and Control (HSCC '20)*, pp. 1–11. ACM. <https://doi.org/10.1145/3365365.3382222>
- [10] Sutton, R. S. (1988). *Learning to predict by the methods of temporal differences*. Machine Learning, 3(1), 9–44.
- [11] Watkins, C. J. C. H., & Dayan, P. (1992). *Q-learning*. Machine Learning, 8(3–4), 279–292.
- [12] Blois, B. (2024). *Vérification de politiques d'apprentissage par renforcement*. Rapport de stage, CNRS@CREATE, Singapour.