# Découverte de Politiques Interprétables pour l'Apprentissage par Renforcement via la Programmation Génétique

Mathurin Videau

▶ To cite this version:

Mathurin Videau. Découverte de Politiques Interprétables pour l'Apprentissage par Renforcement via la Programmation Génétique. Intelligence artificielle [cs.AI]. 2021. hal-03359238

HAL Id: hal-03359238
https://inria.hal.science/hal-03359238v1

Submitted on 30 Sep 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Découverte de Politiques Interprétables pour l'Apprentissage par Renforcement via la Programmation Génétique

Master IASD : Rapport de stage

Mathurin VIDEAU

**Encadrants :** Alessandro LEITE et Marc SCHOENAUER

*INRIA Saclay Île-de-France, Équipe TAU*

23 septembre 2021

**Résumé**

L'apprentissage par renforcement profond a connu un succès remarquable au cours des dernières années pour la résolution d'un large éventail de problèmes de contrôle difficiles. Les milliers de poids et non-linéarité constituant les réseaux de neurones, clef de voûte de cette approche, les rendent cependant incompréhensibles. Le présent rapport présente l'application de la programmation génétique à diverses tâches de contrôle. L'objectif de cette méthode est de produire des politiques symboliques interprétables. Tout d'abord, nous nous intéressons à la viabilité de ces politiques, aussi bien sûr le plan des performances que sur le plan de l'interprétabilité. Puis, nous explorions différentes stratégies pour échapper aux optimums locaux afin d'améliorer leurs performances. Nos résultats montrent que cette approche est une alternative crédible au réseau de neurones pour des tâches concrètes.

## 1 Introduction

Nombre d'avancées importantes en apprentissage automatique ont été réalisées grâce à l'apprentissage profond [1]. Cependant, ces modèles, bien que performants, sont particulièrement opaques. Cette caractéristique de « boîte noire » réduit fortement leurs applications dans les domaines pour lesquels une compréhension fine du modèle est nécessaire comme la santé, l'aéronautique et le spatial, ou la finance. De ce constat, s'est développé le concept d'intelligence artificielle explicable (*eXplainable Artificial Intelligence (XAI) en anglais*). Malheureusement, il n'y a pas de caractérisation claire de l'*explicabilité* d'un algorithme. Le flou autour de cette notion provient en partie du fait qu'une explication peut prendre diverses formes et dépend notamment du public auquel elle s'adresse. Il existe ainsi de nombreuses approches très différentes pour atteindre cet objectif d'explicabilité [2].

L'explicabilité ne permet pas seulement de produire des algorithmes fiables, elle possède bien d'autres vertus. Par exemple, la compréhension des connaissances acquises par le modèle peut permettre de vérifier la présence de biais. In fine, il est

envisageable de tirer parti de ces connaissances pour enrichir notre propre compréhension d'un problème. Même d'un point de vue purement technique, l'explicabilité rend le modèle plus facile à déboguer et permet ainsi de mieux comprendre le processus d'apprentissage qui en découle.

Lors de ce stage, nous nous sommes intéressés à l'explicabilité dans le cadre de l'apprentissage par renforcement (Reinforcement Learning (RL) en anglais) et en particulier aux politiques interprétables pour des problèmes de contrôle. Le terme *interprétable* signifie ici que la politique est intelligible et fait sens pour un être humain [3]. Dans notre cas, les politiques sont représentées sous forme d'expressions mathématiques qui associent un état du système à une action. Idéalement, ces expressions doivent être concises pour être intelligible. Pour ce faire, la recherche de cette politique est ici menée par le biais de la *Programmation Génétique* (GP) [4]. On cherche ici à exploiter la capacité qu'a la programmation génétique à concevoir des programmes dédiés à une tâche pour, d'une part, produire des politiques claires et compréhensibles par l'homme et, d'autre part, pour améliorer l'apprentissage en sélectionnant et en produisant des caractéristiques importantes de l'espace d'état.

Dans un second temps, on s'est intéressé également au potentiel compromis entre interprétabilité et performance : *Est-il possible de remplacer un modèle boîte noire par un modèle interprétable en apprentissage par renforcement ?*

Dans cette optique, ce travail apporte les contributions suivantes :

— Évaluation d'utilisation de la programmation génétique à des tâches de contrôle sur divers environnements et en comparant aux performances des réseaux de neurones (Section 4.1) ;

— Application de la programmation génétique pour des tâches de contrôle continu multidimensionnel et l'analyse de l'interpretabilité des politiques générées (Section 4.2) ;

— Évaluation de l'utilisation de la programmation génétique linéaire (*Linear GP* en anglais) [5] en apprentissage par renforcement (Section 4.1) ;

— Mise en place de deux approches pour échapper aux maximums locaux (Section 5), en l'occurrence *Imitation Learning* (Section 5.1) et *Quality Diversity* (Section 5.2) ;

— Développement d'une stratégie d'évaluation des individus sous forme de bandit (Annexe A).

L'ensemble des politiques peuvent être observées dans leur environnement à l'adresse : https://shortest.link/VHv

# 2 Travaux Connexes

L'explicabilité dans le domaine de l'apprentissage par renforcement (RL) ne s'est développée que très récemment, avec l'avènement de l'apprentissage par renforcement profond (DRL). Il en est de même pour l'interprétabilité qui compte une accélération des publications ces dernières années. Une partie des auteurs utilise des arbres de décision [6–10]. Ces arbres de décision permettent de comprendre la segmentation de l'espace d'état en différentes actions. Bien souvent, le comportement du réseau de neurones est distillé dans l'arbre de décision [7–9].

D'autres auteurs utilisent des politiques ayant la forme de programme ou d'expression symbolique [11–18]. Ces travaux empruntent deux approches différentes : la synthèse de programmes [12, 13] et la régression symbolique [14–18].

Tableau 1 – Approches existantes pour les politiques interprétables sous forme d'expression ou de programmes

| Étude | Approche | Environnement | Objectif | Politique(s) |
|---|---|---|---|---|
| *Maes et al.* [11] | Bandit | Mountain Car<br>Acrobot | Récompense cumulative<br>complexité | Symbolique |
| *Verma et al.* [12] | NN + Bayesian opt | TORCS | Récompense cumulative | Programme |
| *Landajuela et al.* [18] | Deep Symbolic Regression | OpenAI Gym :<br>Classical control, Box2D,<br>Mujocco | Récompense cumulative | Symbolique |
| *Liventsev et al* [13] | RNN + GP | OpenAI Gym (4 envs) | Récompense cumulative | *Brainfuck* |
| *Zhang et al.* [17] | NN + EFS | Acrobot<br>Mountain Car<br>Industrial Benchmark [20] | MSE | Symbolique |
| *Wilson et al.* [16] | Mixed type<br>Cartesian GP | Atari | Récompenses cumulative | Graphe |
| *Kubalík et al.* [14] | Multi-Gene GP | 1-DOF, 2-DOF<br>Magman | Bellman MSE | Symbolique<br>*Value function* |
| *Hein et al.* [15] | GP | Montain Car,<br>CartPole<br>Industrial Benchmark [20] | Récompense cumulative<br>complexité | Symbolique |
| **Cette étude** | **GP + Linear GP** | **OpenAI Gym :<br>Classical control, Box2D,<br>Mujocco** | **Récompense cumulative<br>complexité** | **Symbolique** |

Dans la première approche, des réseaux de neurones sont utilisés pour construire les programmes. Dans *Verma et al.* [12], le programme est conçu à partir d'une base appelée *sketch*. Cette base est modifiée itérativement de manière à réduire l'erreur quadratique entre la sortie du réseau de neurones et le programme. À la fin de l'apprentissage, la politique interprétable possède une meilleure capacité de généralisation que le réseau de neurone originel. *Liventsev et al.* [13] utilisent le langage de programmation *Brainfuck* pour former la politique. Ces politiques sont produites à la fois par la programmation génétique et par un réseau de neurones récurrent (RNN). Les auteurs montrent que cette combinaison écrit de meilleurs programmes. Malheureusement, ces résultats sont nettement en dessous de l'apprentissage profond.

Dans la seconde approche, la majorité des papiers utilise la programmation génétique [14–17]. Historiquement, cette approche a été utilisée en RL pour produire avec succès des régulateurs [19]. Mais, son utilisation en termes d'interprétabilité et sur des environnements plus divers n'est que récente. Par exemple, *Wilson et al.* [16] appliquent ce type de méthodes sur des jeux Atari. Les résultats sont très variables, mais certaines politiques parviennent à faire mieux qu'un réseau de neurones. Néanmoins, ces dernières ne sont pas toujours interprétables à cause de leurs grands nombres d'opérations. *Hein et al.* [15] l'utilisent aussi sur trois environnements de contrôle continue. Une nouvelle fois, les politiques interprétables donnent des résultats proches des réseaux de neurones. Ainsi, les deux stratégies présentées précédemment optimisent leurs politiques en évaluant directement leurs scores au sein de l'environnement. De manière différente, les approches de *Kubalík et al.* [14] visent à obtenir une expression analytique de la *Value function*. Pour l'obtenir, le problème est formulé à l'aide d'un objectif utilisant le point fixe de l'équation de *Bellman*. Néanmoins, il est nécessaire de connaître la fonction transition pour effectuer l'apprentissage.

Très récemment, la régression symbolique à l'aide de réseaux de neurones a été appliquée à divers problèmes de contrôle continu [18]. Les politiques produites sont compétitives voir dépassent les performances obtenues avec un réseau de neurones. De plus, lorsque la dynamique du système est connue, les auteurs prouvent la stabilité des politiques symboliques. Cependant, pour les problèmes ayant des actions multidimensionnelles, cette approche nécessite un réseau de neurones pré-entraîné.

Ainsi, nous constatons que les publications utilisant la programmation génétiques pour des problèmes de contrôle évaluent leurs performances sur quelques environnements spécifiques (Tableau 1). L'un des buts de cette étude est d'estimer les limites de cette méthode en l'appliquant à diverses tâches de contrôle.

# 3 Évolution de Contrôleur sous Forme de Programme

## 3.1 Notations

Le problème d'apprentissage par renforcement est formulé comme un *processus de décision markovien* (PDM) avec un espace d'état $\mathcal{S}$, un espace d'action $\mathcal{A}$, une fonction de transition déterministe $f : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ et une fonction de récompense $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. Par conséquent, à chaque pas de temps discrets $t = 0, 1, 2, \ldots,$ un agent observe à l'instant $t$ un état $s_t \in \mathcal{S}$ et choisit une action $a_t \in \mathcal{A}$. De cette action découle le prochain état $s_{t+1} = f(s_t, a_t)$ et la récompense perçue $r_t = R(s_t, a_t)$. Ainsi, l'objectif de l'agent est de maximiser la récompense cumulative qu'il va obtenir au terme de simulations dans son environnement (Équation (1)).

Soit $\gamma \in [0, 1]$ le facteur d'escompte, $\pi$ une politique, $s_0 \in \mathcal{S}$ l'état initial et un horizon $T \in [1, +\infty]$.

$$\mathcal{R}(s_0, \pi) = \sum_{t=0}^{T-1} \gamma^t R(s_t, \pi(s_t)) \tag{1}$$

avec $s_{t+1} = f(s_t, \pi(s_t))$ et $\pi : \mathcal{S} \mapsto \mathcal{A}$.

Dans ce cas, la meilleure politique est celle qui obtient la plus grande récompense cumulative $\mathcal{R}$ moyenne.

## 3.2 Programmation Génétique

La programmation génétique (GP) est une méthode de synthèse de programme dont le processus d'optimisation est inspiré de l'évolution darwinienne.

### 3.2.1 Processus évolutif

Le processus fait évoluer une population dont les éléments sont à chaque génération évaluées, sélectionnées et modifiées. Les éléments constituant la population sont appelés *individus* ou dans notre cas *programmes*. La première population $P_0$ est initialisée avec des programmes aléatoires. Puis, la population initiale évolue itérativement en suivant trois phases : évaluation, sélection et variation (Figure 1).

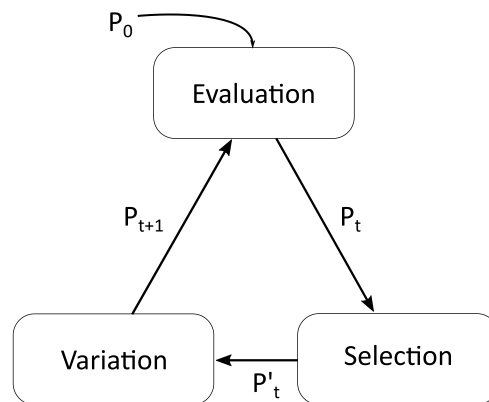



FIGURE 1 – Processus évolutif employé par la programmation génétique

L'**évaluation** attribut un score de « *fitness* » à chaque individu et constitue ainsi l'objectif de l'optimisation. Conformément à l'objectif de performance de la politique, ce score est obtenu par évaluation Monte-Carlo de la récompense cumulative moyenne via $\mathcal{R}$. La simulation étant une action coûteuse en ressources, une stratégie d'évaluation des individus a été mis en place. D'une part, le budget de simulation est augmenté graduellement tout au long de l'évolution.

D'autre part, chaque individu est vu comme le bras d'un bandit ce qui permet de mieux répartir le budget de simulation. Cette stratégie est exposée plus en détail dans l'Annexe A. Aussi, un second objectif décrivant la complexité du programme peut être ajoutée. Ici, à chaque opérateur est attribué un coût (Tableau 2). Ce dernier est minimisé afin d'obtenir les solutions les plus simples et régulières possibles. Cela présente aussi l'avantage de fournir un front de Pareto permettant de choisir un compromis entre complexité et performance d'une solution.

La **sélection** vise à propager le matériel génétique des meilleurs individus à la génération suivante. Ces individus sont appelés *parents*. Le critère de sélection dépend des objectifs et de ce que l'on souhaite promouvoir au sein de la population. Dans notre cas, la sélection par tournois est utilisée en mono-objectif et NSGA–II [21] en multi-objectifs. La méthode de sélection et le nombre de parents $\mu$ sont des paramètres de l'algorithme.

La **Variation** dont le but est de produire de nouveaux individus à partir des parents. Ces nouveaux individus sont appelés *enfants*. Elle est constituée d'un opérateur de *mutation* qui produit des variations locales d'un individu et d'un opérateur de *croisement* qui crée un individu en croisant deux parents. La probabilité de croisement, de mutation et le nombre d'enfants $\lambda$ généré sont des paramètres de l'algorithme.

Tableau 2 – Coût des opérateurs

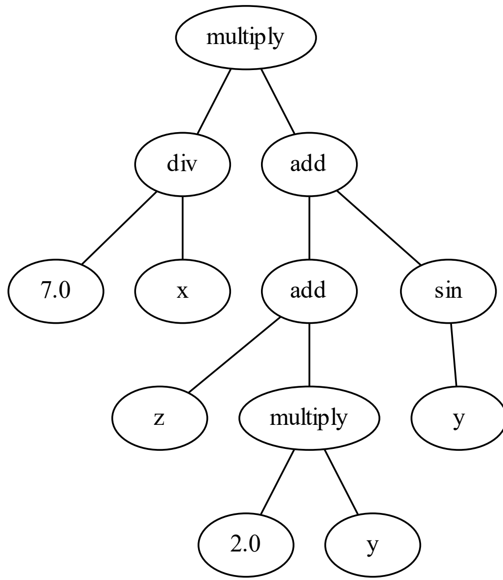
| **Opérateur** | **Complexité** |
|---|---|
| Variables | 1 |
| Constantes | 1 |
| $+, -, \times$ | 1 |
| $/$ | 2 |
| exp, log, ^ | 3 |
| sin, cos | 4 |
| $<, >$ | 4 |
| if | 5 |

### 3.2.2 Représentation des programmes

Pour que les programmes puissent être manipulés lors de l'évolution, ces derniers doivent suivre une même représentation. Cette représentation est interprétée puis exécutée lors de la phase d'évaluation. Ici, deux types de représentation sont évalués : la programmation génétique en arbre (*Tree GP* en anglais) et la programmation génétique linéaire (*Linear GP* en anglais).

La **programmation génétique en arbre** [4] est la représentation la plus classique. Le programme prend la forme d'un arbre (Figure 2 (a)) décrivant les opérations à suivre. Ainsi, une mutation est effectuée en modifiant un sous arbre et un croisement en échangeant une branche. À noter que du fait de sa structure, l'arbre ne possède qu'une seule sortie. Pour l'étendre au cas multidimensionnel, les arbres sont évolués en équipe. De ce fait, les actions se développent indépendamment les une des autres. Afin de prendre en charge nativement le cas multidimensionnel, nous avons utilisé une autre représentation dite «*linéaire* ».

La **programmation génétique linéaire** [5] représente les programmes comme une suite d'instructions (Figure 2 (b)). Une instruction consiste en la manipulation des valeurs du registre. Pour effectuer une opération, une instruction nécessite trois éléments : l'opération, les indices d'entrées et l'indice de sortie. Ainsi une instruction s'interprète comme «*Prendre l'élément en position $x$ du registre, effectuer l'opération et stocker le résultat en position $y$ du registre*». La sortie du programme est donné par le contenu du registre (par exemple en position 0). Afin de pouvoir faire évoluer les programmes, la mutation est définie comme l'ajout, la suppression ou la modification d'une instruction et le croisement comme l'échange d'un segment d'instruction entre deux programmes. Pour encourager l'apparition de programme court, un biais envers la mutation par suppression d'instruction est appliqué. À noter que cette




(a) **Tree GP**

```
float x = ‘valeur de x‘,
      y = ‘valeur de y‘,
      z = ‘valeur de z‘

\\initialisation du registre
float registre[] = [
    0.0, 0.0, 0.0, \\registre de calcule
    x, y, z,       \\registre pour les entrées
    2.0, 7.0 ]     \\registre pour les constantes

void programme(r){
    r[0] = r[6] * r[4]\\ 2 * y
    r[0] = r[5] + r[0]\\ z + (2y) (1)

    r[1] = sin( r[4] )\\ sin(y) (2)
    r[0] = r[0] + r[1]\\  (1) + (2) (3)

    r[2] = r[7] / r[3]\\ 7 / x (4)
    r[0] = r[0] + r[2]\\ (3) + (4)
}
\\execution
programme(register)
\\la sortie est en position 0
float result = register[0]
```

(b) **Linear GP**

FIGURE 2 – l'expression $\frac{7}{x} \times (z + 2y + \sin(y))$ est représentée à gauche par un arbre et à droite de manière linéaire.

représentation peut produire des programmes dont les instructions sont inutiles. Ces instructions ne sont pas reliées à la sortie et sont donc détachées du graphe. Pour contrer cet effet, certaines opérations peuvent être appliquées sur le programme effectif. La transcription du programme en un programme effectif dépend de sa taille $n$ et est effectuée en $\mathcal{O}(n)$.

Par ailleurs, pour chaque représentation, les constantes sont traitées de manière particulière. À la création du programme, elles sont initialisées aléatoirement, puis à chaque génération, une constante peut être mutée de manière aléatoire. Cette mutation consiste en un bruit gaussien ajouté à la constante ou en une réinitialisation aléatoire.

Nous en avons terminé avec les explications concernant la programmation génétique. Les sections suivantes sont dédiées aux expérimentations menées en utilisant ces représentations. L'ensemble de ces expérimentations ont été faites en `python` à l'aide de la librairie `Deap` [22] et `QDpy` [23] pour GP et `stable-baseline3` [24] pour les réseaux de neurones.

# 4 Résultats

Les approches par programmation génétique ont été évalués à l'aide d'*Open AI Gym* [25] sur trois types d'environnements de contrôle différents : *classic control*, *Mujoco* [1] et *Box2D* [2]. Ces environnements ont été divisés en deux parties afin de mieux refléter leurs niveaux de difficultés. En effet, les environnements *classic control*

1. En particulier la version *Bullet* des environnements [26]
2. Des animations de ces environnements sont disponibles sur : `gym.openai.com/envs`

Tableau 3 – Récompense cumulative moyenne

| Environnement | Tree GP | Linear GP | NN | NMCS |
|---|---|---|---|---|
| Cartpole | 500.0 | 500.0 | 500.0† | 484.27 |
| Acrobot | -83.17 | -80.99 | −82.98† | -89.69 |
| MountainCar | 99.31** | 88.16 | 94.56‡ | 97.89 |
| Pendulum | -154.36 | -164.66 | -154.69‡ | -210.71 |
| InvDoublePend | 9092.17 | 9089.50 | 9304.32‡ | – |
| InvPendSwingUp | 893.35 | 887.08 | 891.45‡ | – |
| LunarLander | 287.58 | 262.42 | 269.31† | – |
| BipedalWalker | 268.85 | 257.22 | 299.44* | – |
| BipedalWalkerHardcore | 9.25 | 10.63 | 246.79* | – |
| Hopper | 999.19 | 949.27 | 2604.91‡ | – |

Les réseaux de neurones sont les réseaux de neurones pré-entraînés issu de [30].

† *PPO* [27]

‡ *SAC* [28]

* *A2C* [29]

** Exploite la protection des opérateurs à l'overflow et utilise comme non-linéarité la meilleure politique sans cette exploitation à un score de 97.59.

sont les plus faciles et permettent de vérifier la viabilité de la méthode. Tandis que *Box2D* et *Mujocco* sont plus complexes. L'évaluation menée porte sur deux critères de performance et d'interprétabilité.

## 4.1 Analyse Quantitative

Afin de rendre compte de la compétitivité de la programmation génétique, nous l'avons comparé à deux autres méthodes. La première, état de l'art en RL, utilise des réseaux de neurones pour représenter la politique. En particulier, on retiendra le résultat de la meilleure politique obtenu à l'aide d'une de ces trois méthodes : *PPO* [27], *SAC* [28] et *A2C* [29]. Ces réseaux sont issus du dépôt [30]. La seconde, *Nested Monte-Carlo Search* (NMCS) [31, 32], est une méthode de recherche Monte-Carlo donnant de bons résultats pour optimiser une séquence, donc un arbre comme pour TreeGP.

Commençons par la première moitié du Tableau 3 correspondant aux environnements de contrôle simple. La méthode NMCS obtient les moins bons résultats. Le temps de simulation augmentant pour des tâches plus complexes, l'exécution devient trop longue pour des playouts de niveau 3. Les playouts de niveau 2 donnant des résultats médiocres, la méthode NMCS a été écartée de la suite des expérimentations. Pour les deux autres méthodes utilisant la programmation génétique, les politiques symboliques donnent des résultats similaires à ceux proposés par un réseau de neurones, voire supérieur pour deux des quatre environnements. Intéressons-nous maintenant à la seconde partie du tableau (Tableau 3) portant sur les environnements *Box2D* et *Mujocco*. Ces politiques symboliques donnent de résultats similaires aux réseaux de neurones sur trois des six environnements. Une baisse de performances est particulièrement visible pour les environnements de locomotions. Effectivement, les politiques produites par programmation génétique ont tendance à rester bloquées

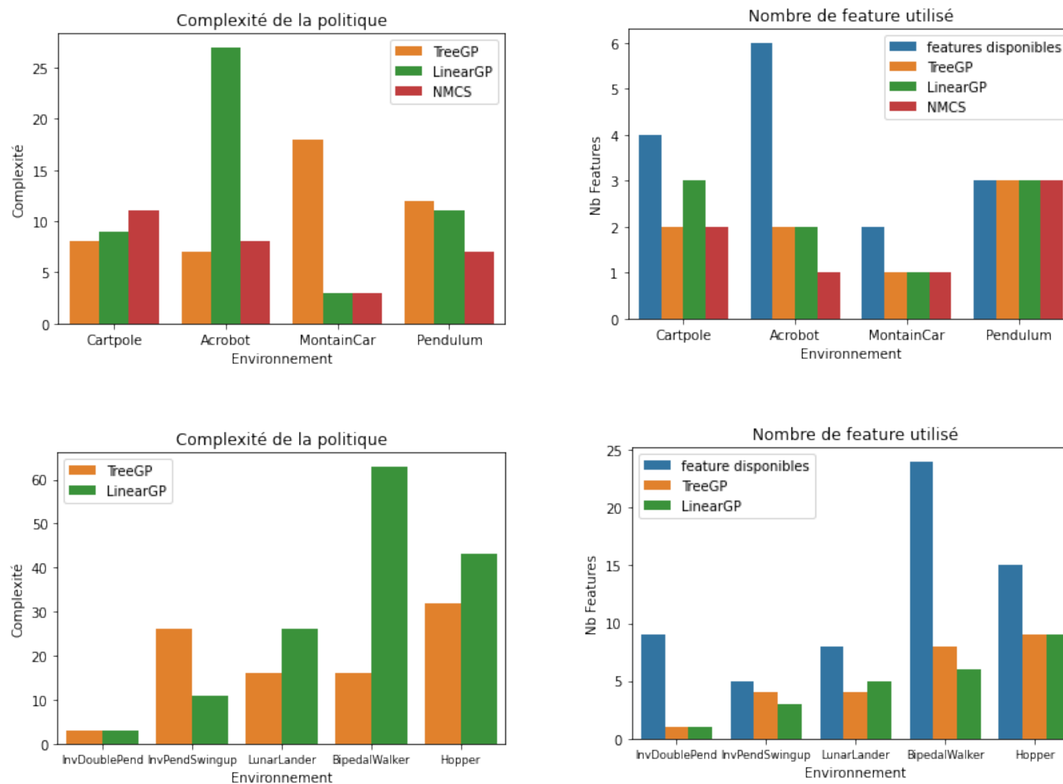



FIGURE 3 – Complexité des politiques (gauche) et features utilisé (droite)

dans des optimums locaux. Dans ces environnements la politique ne parvient pas à effectuer un seul pas et se contente de rester debout le plus efficacement possible. Aussi, les Figures 10 et 11 en Annexe C montrent que les politiques utilisant un réseau de neurones tendent, de manière générale, à avoir une variance légèrement plus faible.

Pour mesurer l'interprétabilité de chacune des méthodes, nous avons utilisé deux critères : la complexité de l'expression (Tableau 2) et le nombre de features utilisés. Ces deux données sont représentées sur la Figure 3. Ainsi, les politiques symboliques ont une expression de complexité faible, inférieur à 65, les rendant facilement lisibles. À titre de comparaison, la complexité d'un réseau de neurones utilisé pour les tâches *classic control* dépasse facilement un score de complexité de 10000. De plus, la programmation génétique semble parvenir à sélectionner les caractéristiques importantes de l'espace d'état. En effet, sur la plupart des tâches, moins de la moitié des features sont utilisées par la politique (Figure 3).

À l'issue de ces travaux mesurant quantitativement les politiques symboliques, nous allons à présent étudier leurs interprétabilité de manière plus qualitative.

## 4.2 Analyse de l'Interprétabilité

Cette section vise à analyser l'interprétabilité des politiques symboliques. Dans l'idéal, à la lecture de la politique, un être humain doit être capable de comprendre/simuler le comportement de la politique. Une telle politique est dite *simulable*. On tentera ici de montrer l'interprétabilité des politiques au travers de deux exemples

*LunarLander et BipedalWalker.* D'autres exemples d'expression sont apportés en Annexe D.

Pour l'environnement *LunarLander*, la politique a pour expression :

$$\text{main engine} : a_0 = \underbrace{(y > 0)}_{(1)} ? \underbrace{(-0.37y - \dot{y} + 0.1)}_{(2)} : 0$$

$$\text{side engine} : a_1 = (\underbrace{4(\theta - x)}_{(3)} \underbrace{-\dot{x})4}_{(4)}$$

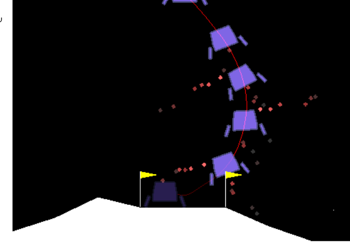


FIGURE 4 – Trajectoire résultant de la politique

Ainsi, l'expression peut-être décomposée en différent blocs fonctionnels :

(1) Stop le moteur après l'atterrissage.
(2) Réduit graduellement la vitesse du module à l'approche de la plateforme.
(3) Stabilise la fusée à la verticale et la maintient autour du centre.
(4) Réduit la vitesse de la fusée proche du centre.

Dans ce cas précis, la politique est très intuitive. Cette compréhension permet même d'aller plus loin et d'y voir quelques défauts. Par exemple, le moteur principal ne prend pas en compte l'orientation de la fusée. Par conséquent, si la fusée est à l'envers, celle-ci va continuellement accélérer vers le sol jusqu'à fatalement s'écraser. Un deuxième défaut visible est que la vitesse angulaire de la fusée est ignorée. Ainsi, lorsque l'appareil est soumis à une forte rotation, la politique ne parvient pas à redresser sa trajectoire. Ces hypothèses ont pu être vérifiées expérimentalement et constituent des cas limites d'utilisation de la politique. Pour corriger ces défauts, les cas limites peuvent être intégrés à la simulation et le processus évolutif est relancé depuis la dernière génération ou alors, les expressions peuvent être corrigées manuellement.

Intéressons-nous maintenant à une politique moins favorable à l'interprétation.

Pour l'environnement *BipedalWalker*, la politique a la forme suivante :

$$\text{hip1} : a_0 = \text{knee1}.\theta$$

$$\text{knee1} : a_1 = \frac{\text{knee1}.\dot{\theta}}{\text{lidar}_5}$$

$$\text{hip2} : a_2 = \frac{\text{lidar}_7}{\text{knee1}.\dot{\theta}} - \text{hip2}.\theta + \text{hull}.\theta$$

$$\text{knee2} : a_3 = \text{hull}.\theta - \text{knee2}.\theta$$

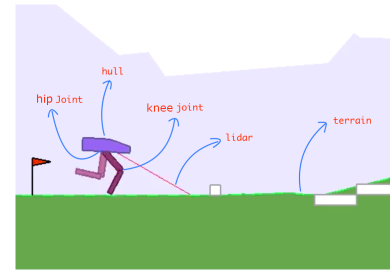



FIGURE 5 – BipedalWalker environemment

Ici, l'objectif étant plus abstrait, la politique est bien moins facile à comprendre. En l'occurrence, il paraît difficile de percevoir le comportement adopté par l'agent à la simple vue de la politique. Néanmoins, certaines informations peuvent être extraites. Tout d'abord, on remarque que de manière générale les actions dépendent peu du *lidar* et beaucoup plus de l'état des articulations. Cette dernière information permet notamment de mieux comprendre comment les actions sont synchronisées. Ainsi, la plupart des actions dépendent de leur propre articulation et d'une autre. Par exemple,

les articulations de la première jambe dépendent uniquement de l'articulation *knee1*. Alors que la seconde jambe, elle, utilise des informations appartenant au *hull* ce qui laisse penser quel est responsable du maintien de l'équilibre de l'agent et sert donc de jambe d'appui.

Comme vu précédemment, les politiques symboliques ne sont pas toutes aussi faciles à interpréter. De manière générale, plus la politique porte sur un domaine à l'objectif abstrait, moins elle devient intuitive. Néanmoins, ces politiques, grâce à leur représentation mathématique, restent facilement analysable dans la mesure où leurs expressions gardent une taille raisonnable.

# 5 Échapper aux Optimums Locaux

Comme vu précédemment, la programmation génétique a tendance à rester bloquée dans des optimums locaux. Ce problème est en partie dû à l'espace de recherche discontinu. En effet, une modification d'une seule des opérations induit de grands changements sur les sorties du programme. Cela à notamment pour effet de rendre difficile la transition d'un optimum à un autre. Ainsi, la phase d'exploration s'en trouve diminuée. Dans les deux sections qui suivent, nous essayerons de mettre en place des stratégies pour éviter ces optimums locaux et améliorer les performances des politiques symboliques.

## 5.1 Imitation Learning

L'*imitation learning* est une approche dont le but est de faire reproduire le comportement d'un expert par un agent. Dans notre contexte, un réseau de neurones pré-entraîné joue le rôle de l'expert et un programme a pour but de l'imiter. Une manière simple de répondre à ce problème est de constituer un jeu de données avec les trajectoires de l'expert. Dans ce jeu de données, chaque état est associé à une action $\mathcal{D} = \{(s_0, a_0), \ldots, (s_n, a_n)\}$ avec $s, a \in \mathcal{S} \times \mathbf{A}$ et $n \in \mathbb{N}$. Le problème d'apprentissage devient alors supervisé et le processus évolutif a pour objectif la minimisation de l'erreur $\ell(\pi(s), a)$. Les actions étant continues, la fonction de coût $\ell$ est l'erreur quadratique moyenne. Pour des politiques symboliques [15, 17, 18], ces approches donnent des résultats satisfaisants seulement pour des environnements simples et ne passent pas à l'échelle. L'écart entre l'objectif supervisé et l'objectif d'apprentissage par renforcement semble trop important.

### 5.1.1 *DAgger*

Afin de réduire cet écart, le jeu de données peut être dynamique comme le propose la méthode *DAgger* [33]. En effet, à intervalles réguliers, les trajectoires des meilleurs programmes sont comparées à l'action prise par l'expert. Quand les deux actions divergent de manière trop importante, l'action de l'expert est ajoutée au jeu de donnée. En procédant ainsi, les programmes peuvent mieux explorer l'environnement. Malgré cette modification, la politique reste médiocre. De plus, les solutions sont beaucoup moins interprétables à cause de leur taille et complexité (Figure 6).

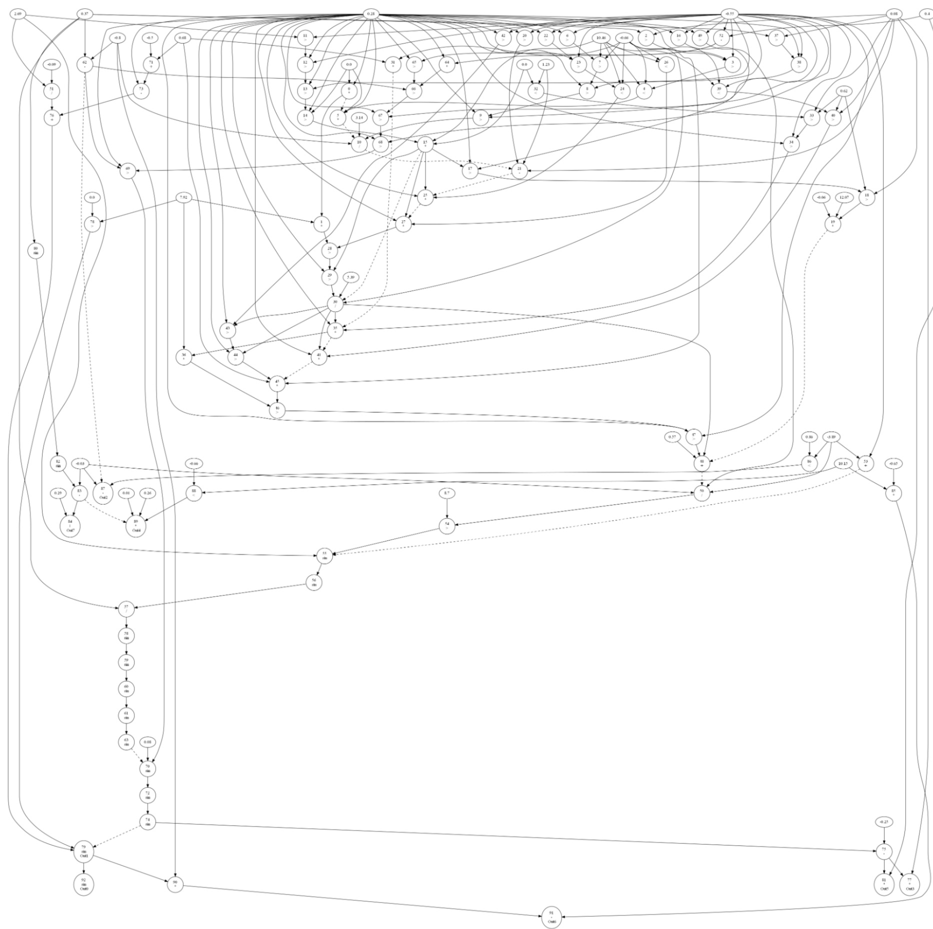


FIGURE 6 – Exemple de politique symbolique inintelligible

### 5.1.2 L'évolution guidée par un expert

L'objectif de supervision seul ne donnant pas de résultat probant, nous avons ajouté l'objectif de supervision à l'objectif de récompense moyenne présenté précédemment. Ce nouvel objectif pousse le processus évolutif à suivre la solution de l'expert, et ainsi éviter les optimums locaux. Encore une fois, les performances de la politique s'avèrent à peine meilleurs. Néanmoins, la politique parvient à se déplacer, mais celle-ci manque de consistance et de stabilité.

L'approches par *imitation learning* ne parvenant pas à transférer les performances du réseau de neurones au programme évolué, la suite du stage s'est concentrée sur une stratégie différente.

## 5.2 Quality Diversity

*Quality Diversity* (QD) est une méthode d'optimisation récente dont le but est de trouver des solutions à la fois diverses et de qualité. Dans notre cas, cela signifie que l'on souhaite conserver au sien de la population des individus ayant de bonnes performances et des comportements différents.

### 5.2.1 MAP-Elites

Un des algorithmes les plus populaires de ce domaine est *Map-Elites* [34]. Pour maintenir la diversité, chaque individu est placé sur une grille. La cellule qu'occupera l'individu est déterminé par un descripteur représentant son comportement. De cette façon, seuls les individus ayant le même comportement sont mis en compétition. La programmation génétique suit le même cours que présenté à la Section 3.2.1 au détail près que la population est représentée par une grille. Afin de remplir un maximum de cellules, à chaque génération une case est sélectionnée aléatoirement. Le temps de convergence de l'algorithme est donc conditionné par la taille de la grille.

Du fait de l'évaluation Monte-Carlo des individus, nous utilisons une variante de l'algorithme *Map-Elites* plus robuste au bruit [35]. Aussi à la fin du processus d'optimisation *Map-Elites*, quelques générations d'ajustement sont effectuées avec *NSGA-II* pour améliorer et simplifier la politique.

### 5.2.2 Résultats

Les expérimentations ont été menées sur deux environnements : *BipedalWalker* et *Hopper*. Le comportement des individus est représenté par l'angle moyen *hip1* et *hip2* dans le premier environnement et par l'angle moyen *leg*, *foot*, la proportion de temps en contact avec le sol pour le second.

Le Tableau 4 permet de constater une amélioration des politiques pour tous les environnements. La tâche de locomotion *Hopper* obtient le gain de performance le plus significatif. Effectivement, ici, l'agent passe d'une politique statique à une politique de marche. Aussi, la Figure 7 tend à montrer que les politiques découvertes via *Map-Elites* restent interprétables. De plus, elles ont un degré de complexité et un nombre de features similaires à l'approche classique. Ces politiques sont consultables à l'Annexe D.

Tableau 4 – Performances des politiques obtenues via *Map-Elites*

| Environnement | QD-Tree GP | QD-Linear GP | Tree GP | Linear GP | NN |
|---|---|---|---|---|---|
| BipedalWalker | 281.47 | 299.64 | 268.85 | 257.22 | 299.44* |
| Hopper | 2152.19 | 1450.11 | 999.19 | 949.27 | 2604.91‡ |

Les réseaux de neurones sont les réseaux de neurones pré-entraînés issu de [30].
* *A2C* [29]
‡ *SAC* [28]

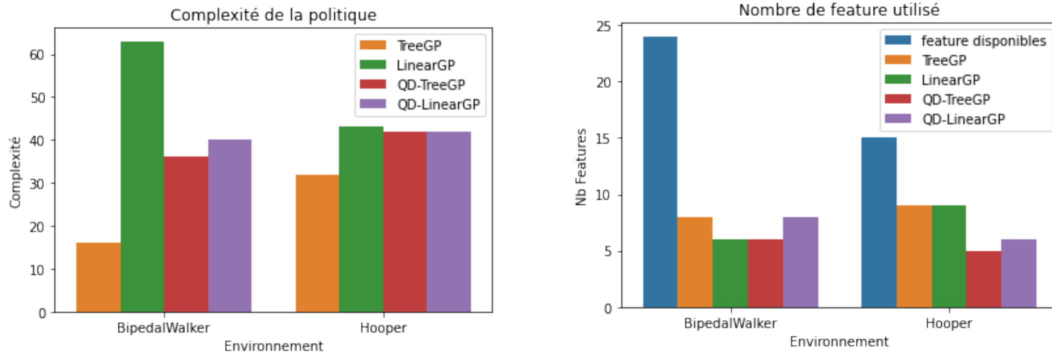



Figure 7 – Complexité des politiques (gauche) et features utilisés (droite)

Par ailleurs, la population sous forme de grille permet d'observer la diversité de solutions découvertes lors de l'évolution. Ces différents comportements pris indépendamment sont des optimums locaux dans lequel aurait pu rester bloqué le processus d'évolution classique. Une analyse plus détaillée de la grille de programme est disponible à l'Annexe E.

# 6 Discussions

Tout d'abord intéressons-nous aux avantages des politiques symboliques produites au-devant du réseau de neurones classique. La première, sujet de cette étude, est l'interprétabilité. Il est souvent possible de tirer des enseignements d'une expression mathématiques concise. De plus, leurs sorties sont beaucoup plus régulières que celle d'un réseau de neurones ce qui les rend plus facilement vérifiables, et donc moins dangereuses à déployer. Aussi, du fait de leur simplicité, ces politiques ont une grande portabilité et peuvent donc être utilisées dans des contextes aux ressources limitées ou nécessitant une faible latence. Enfin, leurs performances sont souvent compétitives face au réseau de neurones. Cet aspect est particulièrement visible pour des tâches dont l'objectif est concret, soit directement en rapport avec l'espace d'états. En outre, cet objectif concret rend l'interprétation de ces politiques plus intuitifs. L'utilisation de la programmation génétique présente un autre avantage. Contrairement aux autres approches, elle retourne un ensemble de solutions. Par conséquent, à la fin du processus évolutif, il est possible de choisir une solution selon les contraintes du problème ou les préférences de l'expert par exemple.

Comparons maintenant GP et Linear GP. Linear GP a été introduit dans cette

étude afin de mieux répondre aux tâches aux actions multidimensionnelles. Malheureusement, dans ce contexte, il ne semble pas y avoir de gain. En particulier, son utilisation n'a pas produit de features communs qui puissent être partagés entre les différentes sorties. De plus, l'espace de recherche est plus complexe et demande l'ajustement de plus d'hyperparamètres que la programmation génétique en arbre. Aussi, la représentation linéaire produit plus facilement des programmes avec des conditions («*if*») imbriquées.

Lors de ce stage, nous avons aussi essayé, comme rapporté plus haut, de mettre en place des stratégies pour échapper aux optimums locaux. Ce problème est particulièrement visible pour des objectifs plus abstraits comme les tâches de locomotion. L'utilisation d'un réseau de neurones pré-entraîné n'a pas abouti à de bons résultats. Les sorties du réseau paraissent trop irrégulières. De plus, ces réseaux peuvent aussi avoir des micro-comportements qui semblent difficiles à transmettre, en l'état, au programme. La Figure 6 en est un exemple, les opérations semblent être ajoutées au fur et à mesure sans structure et sans amélioration significative des performances. Ainsi, le programme construit par programmation génétique semble manqué d'abstraction. Pour obtenir de meilleurs résultats, nous nous sommes tournés vers une autre approche. En l'occurrence, *Map-Elites* permet une meilleure exploration en parvenant à maintenir une diversité de comportement. Néanmoins, la qualité des solutions dépend de descripteurs définis manuellement et de la taille de ses descripteurs. En effet, la grille augmente de manière exponentielle avec la taille des descripteurs ce qui ralentie dramatiquement la convergence de l'algorithme. Très rapidement le procédé devient inutilisable pour des problèmes de plus grande dimension.

Les différents points soulevaient précédemment peuvent donner lieu à des études supplémentaires. La prochaine section vise à suggérer quelques pistes de recherches.

# 7 Développements Possibles

Idéalement, les futurs travaux devront porter sur deux fronts : l'interprétabilité et la performance

La première voie d'amélioration, en lien direct avec ce stage, est de poursuivre les expérimentations sur l'algorithme *Map-Elites*. Dans un premier temps, une amélioration simple pour accélérer le processus d'évolution serait de détecter les programmes fonctionnellement équivalents pour directement leur attribuer la *fitness* retenue en *cache*, et ainsi réduire le nombre de simulations. La détection de fonction équivalente (FEC) peut se faire de manière similaire à [36] en comparant les sorties des programmes avec un ensemble de points. Ce procédé a déjà été utilisé dans l'Annexe E. Dans un second temps, on peut accélérer la convergence de l'algorithme. Par exemple dans la littérature, l'algorithme *SAIL* [37] utilise l'optimisation bayésienne pour sélectionner les cellules à explorer. Un autre moyen, serait d'éviter l'augmentation trop rapide de la grille. Cette augmentation vient du produit cartésien de chaque descripteur. À la manière du *fitness shaping*, il est peut-être possible de pré-entraîner une population sur une grille de taille limitée à quelques descripteurs cibles puis répéter séquentiellement le procédé sur chacun des autres descripteurs d'intérêt restants. De cette manière, on peut espérer tirer parti de la diversité de la grille et des différents descripteurs pour un coût moindre.

Un second axe de recherche plus ambitieux serait de travailler sur la modularité des programmes en permettant la réutilisation de code. Idéalement, la modularité a

le potentiel pour rendre la politique, d'une part, plus performante en augmentant son degré d'abstraction, et d'autre part, plus compréhensibles en jouant le rôle de prototypes en relation directe avec l'objectif. Néanmoins, ce double aspect paraît très difficile à atteindre au regard de la littérature actuel. Une piste possible à suivre, dans le domaine de la modularité, est le *Tangled Graph Program* (TGP) [38]. Cette approche obtient notamment de très bons résultats dans le domaine de l'apprentissage par renforcement visuel. Malheureusement, elle ne prend pas encore en charge nativement les actions continues.

# Conclusion

Au terme de ces cinq mois de stage, la programmation génétique a été employée avec succès à la découverte de politiques symboliques interprétables pour des tâches de contrôle. Les politiques que nous avons générées présentent l'avantage d'être facilement interprétables, portables, transparentes et reproductibles. De plus, nous les avons évaluées sur divers environnements dont certains comportent des actions multidimensionnelles. Les résultats montrent la compétitivité de ces politiques face à l'apprentissage par renforcement profond. Néanmoins, pour des tâches plus abstraites, ce stage a mis en lumière l'importance de l'exploration pour échapper aux optimums locaux. En particulier, l'algorithme *Map-Elites* a permis d'améliorer les performances des politiques sans pour autant en augmenter la complexité. Tous ces éléments mis bout à bout suggèrent que les politiques symboliques peuvent être une alternative aux politiques basées sur des réseaux de neurones. Typiquement, elles s'intègrent parfaitement dans des contextes pour lesquels l'interprétabilité est un critère important ou dans lesquels les ressources sont limitées.

# Remerciements

Je souhaite remercier tout particulièrement mes encadrants, Alessandro Leite et Marc Schoenauer, pour les suggestions et l'aide qu'ils m'ont apporté tout au long du stage. Je voudrais aussi remercier le groupe de travail *HUMANIA* pour leurs présentations inspirantes et leurs suggestions.

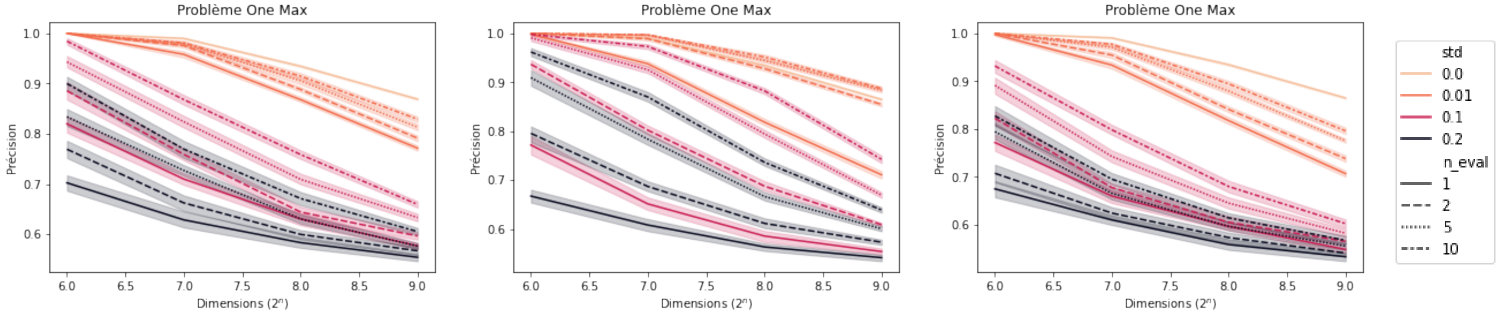



FIGURE 8 – Expérimentation sur un problème simple avec un bruit gaussien. Les graphes correspondent au résultat moyen obtenu sur 25 exécutions.
De gauche à droite : $(\mu, \lambda)$, UCB$(\mu + \lambda)$, $(\mu + \lambda)$.

# A Stratégie d'Évaluation

**Principe :** Afin de mieux répartir le budget de simulation entre les individus, ces derniers sont traités comme les bras d'un problème de bandit. Ainsi, à chaque génération :

1. La *fitness* est donnée par UCB : $\bar{x} + c\sqrt{\frac{\ln(t)}{n}}$ avec $\bar{x}$ le score moyen, $c$ la constante d'exploration, $t$ le budget de simulation et $n$ le nombre d'évaluations de l'individu.
2. Les $\lambda$ nouveaux individus ont une *fitness* de $\infty$, ils sont donc tous évalués au moins une fois.
3. Tant que le budget $t$ n'est pas consommé, on effectue une simulation sur les $k$ meilleurs individus en parallèle. Leur *fitness* est mis à jour et le budget est décrémenté d'une unité.
4. Dans un contexte multi-objectif, les $k$ meilleurs individus peuvent être sélectionnés à l'aide de NSGA-II.
5. Les $\mu$ parents sélectionnés pour la génération suivante voient leur budget de simulation augmenté de $t$, la borne supérieure est augmentée en conséquence : $c\sqrt{\frac{\ln(n'+t)}{n}}$ avec $n'$ le nombre de fois qu'a été tiré le bras durant les générations précédentes.

**Expérimentations :** Pour mieux rendre compte des effets de cette stratégie, nous l'avons testé dans deux contextes différents et contre deux algorithmes. En particulier, le schéma $(\mu, \lambda)$ et $(\mu+\lambda)$ où dans le premier la sélection porte uniquement sur les $\lambda$ enfants (les nouveaux individus) et dans la seconde, la phase de sélection inclut les $\mu$ parents et les $\lambda$ enfants.

La première expérience est effectuée sur le problème artificiel *One Max*. Ce problème d'optimisation porte sur la maximisation du nombre de 1 dans un vecteur de taille fixe $N \in \mathbb{N}$. Pour rendre le problème stochastique, un bruit gaussien $\mathcal{N}(0, \sigma), \sigma \in \mathbb{R}^+$ est ajouté à l'objectif :

$$f(\vec{x}) = \frac{\sum_{i=0}^{N} x_i}{N} + \mathcal{N}(0, \sigma)$$

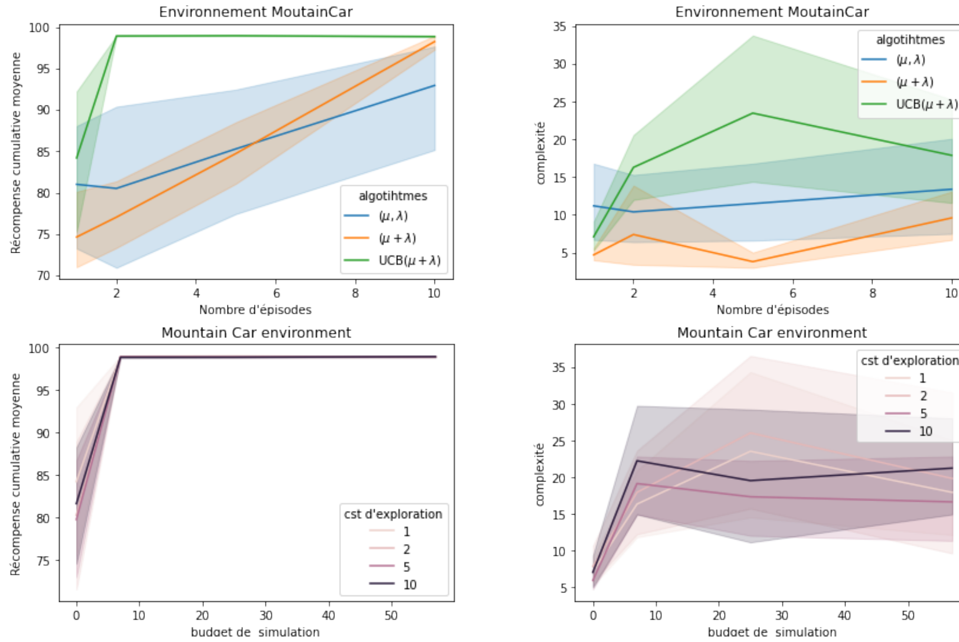



FIGURE 9 – Expérimentation sur l'environnement *Mountain Car*. Les résultats présentés sont moyennés sur 10 exécutions de chaque algorithme. La récompense cumulative maximale ne peut pas dépasser 100.

Dans cette expérience, la difficulté du problème est contrôlé par $N$ et le bruit par $\sigma$. Pour mesurer l'efficacité des différentes approches, $(\mu, \lambda)$ et $\mu + \lambda$ utilisent un budget de $n$ évaluations par individu et un budget équivalent $t$ est calculé pour le bandit. La constante d'exploration est ajustée en accord avec le bruit $\sigma$. La figure 8 montre les résultats de l'expérience :

— De manière générale, la qualité des solutions diminue avec l'augmentation du bruit et la taille du vecteur.
— $(\mu, \lambda)$ donne de meilleurs résultats que $(\mu + \lambda)$ mais cet écart diminue avec la force du bruit et la difficulté du problème.
— La stratégie avec bandit obtient les meilleurs résultats pour toutes les configurations. De plus, pour un bruit faible ($\sigma = 0.01$) et un budget suffisant, elle donne des résultats équivalent à $(\mu, \lambda)$ sans bruit.
— Le budget est mieux utilisé par la stratégie avec bandit. En l'occurrence, l'écart entre les courbes ayant des budgets différent est plus grand que pour les autres stratégies.

La seconde expérience utilise l'environnement *Mountain Car* et correspond mieux à notre problème. Les trois méthodes sont comparées en termes de nombres d'épisodes utilisés pour évaluer une solution. Encore une fois, un budget équivalent est attribué au bandit.

Les résultats présentées Figure 9 montrent que la stratégie avec bandits donne de meilleures performances au regard du budget de simulation. En particulier, la variance est bien plus faible. Le schéma $(\mu, \lambda)$ a une variance beaucoup plus élevée et de moins bons résultats, car il ne conserve pas les meilleurs individus entre chaque génération. C'est pourquoi, pour notre problème, nous avons utilisé le schéma $\mu + \lambda$. Aussi, les graphes tendent à montrer que le choix de la constante d'exploration n'a que peu d'effet sur un problème aussi simple que *Mountain Car*. En revanche, cet effet est moins clair sur la complexité. Malheureusement, il est difficile de conclure

au vu des incertitudes en jeu.

# B Paramètres d'Évolution

Tableau 5 – Paramètres d'évolutions de base pour l'environnement *classical control*

| Représentation | Tree GP | Linear GP |
|---|---|---|
| Sélection | NSGA-II | tournois |
| Ensemble d'opérations | $\{+, -, \times, /, \text{if}\}$ | $\{+, -, \times, /, \text{if}\}$ |
| Paramètre d'évolution | $P_{\text{mut}} = 1.0$ | $P_{\text{mut}} = 1.0$ |
| | $P_{\text{crossover}} = 0.0$ | $P_{\text{ins}} = 0.3, P_{\text{del}} = 0.6$ |
| | | $P_{\text{Instruction_mut}} = 0.5$ |
| | | $P_{\text{crossover}} = 0.0$ |
| Parents $\mu$ | 100 | 100 |
| Enfant $\lambda$ | 100 | 100 |
| Nombre de générations | 500 | 500 |

Tableau 6 – Paramètres d'évolutions de base pour les environnements *Mujocco* et *Box2D*

| Représentation | Tree GP | Linear GP |
|---|---|---|
| Sélection | NSGA-II | tournois |
| Ensemble d'opérations | $\{+, -, \times, /, \text{if}, \exp, \log, \sin\}$ | $\{+, -, \times, /, \text{if}, \exp, \log, \sin\}$ |
| Paramètre d'évolution | $P_{\text{mut}} = 1.0$ | $P_{\text{mut}} = 1.0$ |
| | $P_{\text{crossover}} = 0.0$ | $P_{\text{ins}} = 0.3, P_{\text{del}} = 0.6$ |
| | | $P_{\text{Instruction_mut}} = 0.5$ |
| | | $P_{\text{crossover}} = 0.0$ |
| Parents $\mu$ | 500 | 500 |
| Enfant $\lambda$ | 500 | 500 |
| Nombre de générations | 2000 | 2000 |

# C Récompenses Cumulative

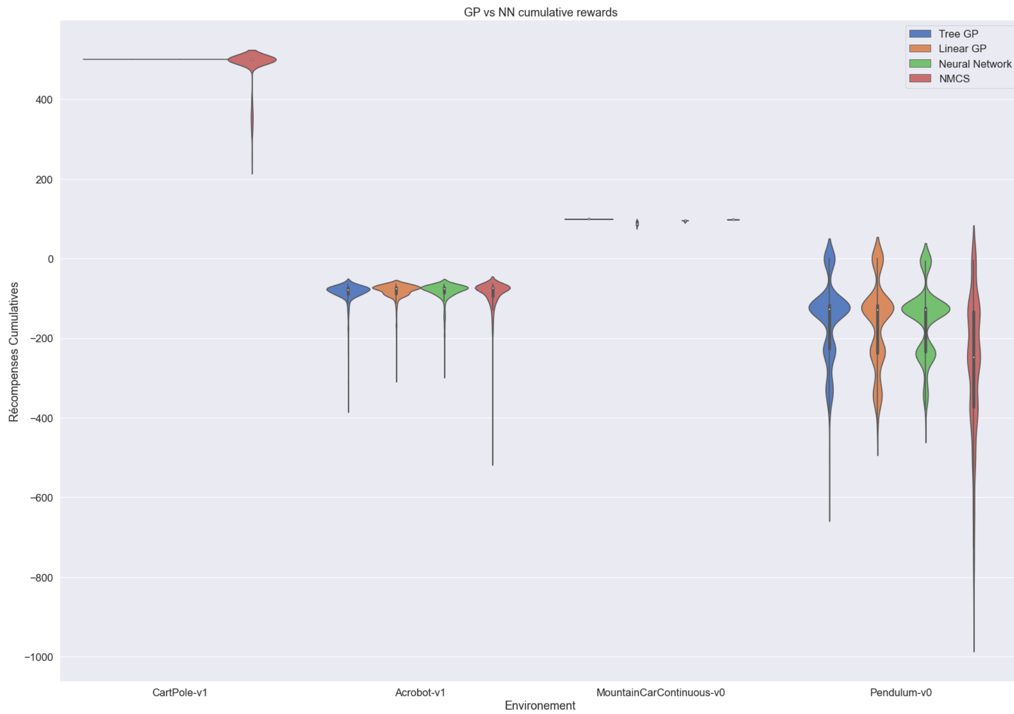



FIGURE 10 – Distribution des récompenses des différentes politiques. Chaque politique est simulée mille fois

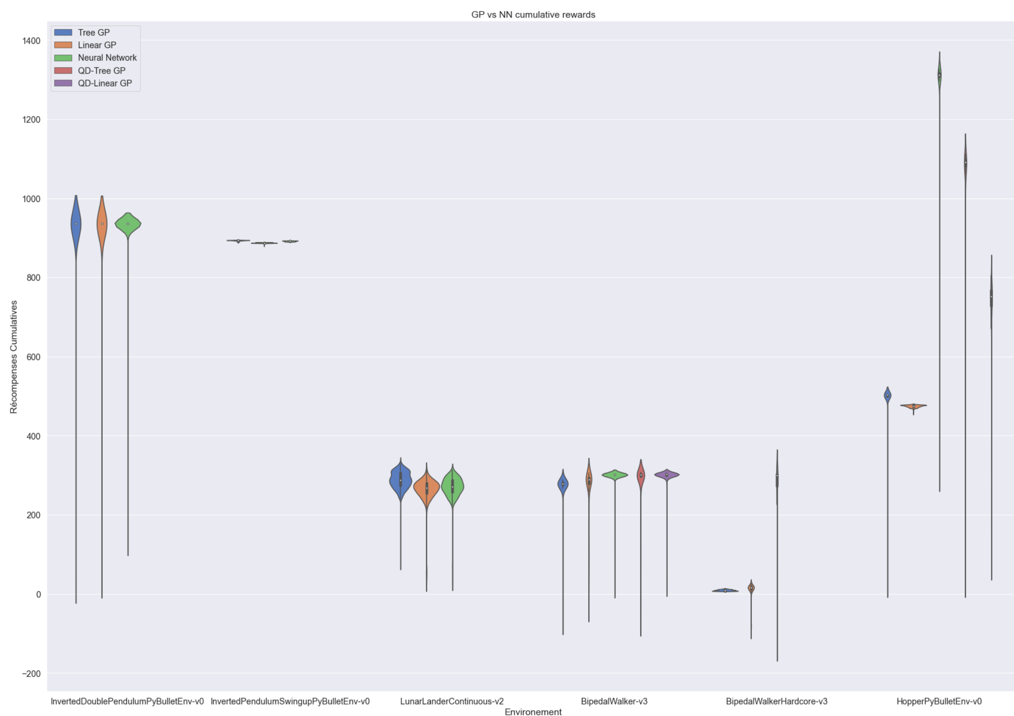



FIGURE 11 – Distribution des récompenses des différentes politiques. Chaque politique est simulée mille fois. Les récompenses du double pendule sont divisées par dix et celles de *Hopper* par deux pour rendre le graphe plus lisible.

## D Expression des Politiques Symboliques

Tableau 7 – Meilleures expressions symboliques générées par chaque environnement

| Environnement | Tree GP | Linear GP | NMCS |
|---|---|---|---|
| CartPole | $\dot{\theta} > -8.44\theta$ | $\dot{\theta} > -\theta - \dot{x}$ | $\frac{8.0}{\dot{\theta}+\theta} > \theta$ |
| Acrobot | $\arg\max(\dot{\theta}_1, \dot{\theta}_1, \dot{\theta}_2)$ | $\arg\max(0.0, (5.8 > \dot{\theta}_2)?\dot{\theta}_2 : -\dot{\theta}_2, (\dot{\theta}_1 < \dot{\theta}_2)?\dot{\theta}_2 + 5.8 : 0)$ | $\arg\max(1, 0, \frac{9.0}{\dot{\theta}_2})$ |
| MountainCar | $((\lvert\dot{x}\rvert > 0.001)?4.86 : 120.22)\dot{x}$ * | $32.6\dot{x}$ | $24\dot{x}$ |
| Pendulum | $\cos(\theta) - \frac{9.16\dot{\theta}+40,14\sin(\theta)}{\cos(\theta)}$ | $0.14 - 4.05\cos(\theta)(4.05\sin(\theta) + \dot{\theta})$ | $\sin(\theta) - 9.0\frac{\cos(\theta)}{\sin(\theta)+\dot{\theta}}$ |

$(\bullet)?(1) : (2)$ correspond à l'opérateur ternaire soit *if*$(\bullet)$ *then* $\{(1)\}$ *else* $\{(2)\}$
* Autre solution sans la non-linéarité : $(-36.17\dot{x} + 11.14)\dot{x}$

Tableau 8 – Expressions complexes produite pour chaque environnement

| Environnement | Tree GP | Linear GP |
|---|---|---|
| InvDoublePend | $-10.7\sin\theta_2$ | $-11.4\sin\theta_2$ |
| InvPendSwingup | $\dot{\theta} + 6.6\sin\theta - \cos\theta + \exp((\sin\theta > -0.76)?(\dot{x}) : (15\dot{x}))$ | $4.24 \times (9.45\sin\theta + 2\dot{\theta} + \dot{x})$ |
| LunarLander | $a_0 = -(2\dot{x} + x) - 0.2$<br>$a_1 = -19.77(\dot{y} - \dot{\theta}(x + 1))$ | $a_0 = (y > 0)?(-0.37y - \dot{y} + 0.1) : 0$<br>$a_1 = 4(4(\theta - x) - \dot{x})$ |
| BipedalWalker | $a_0 = \text{knee1}.\theta$<br>$a_1 = \frac{\text{knee1}.\dot{\theta}}{\text{lidar}_5}$<br>$a_2 = \frac{\text{lidar}_7}{\text{knee1}.\dot{\theta}} - \text{hip2}.\theta + \text{hull}.\theta$<br>$a_3 = \text{hull}.\theta - \text{knee2}.\theta$ | $a_0 = (\dot{y} < \text{hull}.\theta)?(\text{lidar}_0 \times \text{lidar}_6) : 0$<br>$a_1 = (\dot{x} > \text{knee2}.\theta)?\sin(\frac{\text{hip2}.\theta}{19.89}) :$<br>$(\text{knee2}.\dot{\theta} \times \text{knee1}.\theta)$<br>$a_2 = \text{lidar}_3 \times \text{knee1}.\theta$<br>$a_3 = (\text{lidar}_4 < \text{knee1}.\dot{\theta})?1 :$<br>$(\text{lidar}_3\text{knee1}.\theta > 0)?\frac{\text{knee2}.\dot{\theta}}{\text{lidar}_2} : 0$ |
| Hopper | $a_0 = \frac{(s_9+s_7)}{(-25.55s_5-11.69)}$<br>$a_1 = s_2 - s_{10} + (\sin(s_{13}) - s_{12})s_{14} - \log(s_2) - s_{11}$<br>$a_2 = -3.18s_{12}$ | $a_0 = (s_{13} > 0)?0.61 : 0$<br>$a_1 = s_2 - s_{10} - s_{11}$<br>$a_2 = (s_0 - s_3 < s_{12})?s_0 - s_6 - s_3 - s_7 : -3.79s_3 - s_7$ |

$(\bullet)?(1) : (2)$ correspond à l'opérateur ternaire soit *if*$(\bullet)$ *then* $\{(1)\}$ *else* $\{(2)\}$

Tableau 9 – Politique obtenue via *Map-Elite*

| Environnement | QD-Tree GP | QD-Linear GP |
|---|---|---|
| BipedalWalker | $a_0 = 0.31\dot{y}$<br>$a_1 = \sin[((\dot{x}^2 + 0.2)^3 + 5.42\text{lidar}_0)^2]$<br>$a_2 = (-5.34(\text{knee1}.\theta \times \text{lidar}_7))^3$<br>$a_3 = (\frac{\text{lidar}_7}{[(-16.61\sqrt{\text{knee2}.\theta})\sin(\text{knee2}.\theta)]^3})^2$ | $a_0 = (\text{knee1}.\dot{\theta} > -0.62)?\frac{\text{hip1}.\theta}{-0.62} :$<br>$(\text{lidar}_4 < \text{hip1}.\dot{\theta})?(\dot{x} \times \text{lidar}_6) : 0$<br>$a_1 = \frac{\text{lidar}_6}{\text{hull}.\dot{\theta}}$<br>$a_2 = \text{lidar}_4 - \text{knee2}.\theta$<br>$a_3 = 0.3 - \text{knee2}.\theta$ |
| Hooper | $a_0 = \sin(\exp(s_8))$<br>$a_1 = -6.26(s_7 + \sin(s_3 + s_7))$<br>$a_2 = \sin(\sin(s_7) - \sin(s_8) - s_{10}(s_1 - \log(s_8 s_3)))$ | $a_0 = 0.36$<br>$a_1 = ((s_{11} > 0)?\{(s_0 < 0)?\sin(s_7) : 0\} : s_{12} - s_{10}) - s_{11}$<br>$a_2 = (s_0 > s_7)?0.35 : -4.12s_3$ |

$^\sharp$Malheureusement, les sens des états du *Hopper* n'est que connu partiellement.

# E Analyse de la Grille de Programmes

Dans cette annexe, on s'interroge sur la manière dont les programmes s'arrangent dans la grille. En particulier, les programmes suivent-ils un modèle ou sont-ils spécialisés à un unique comportement ? Pour répondre à cette question, nous avons étudié la grille retournée par *Map-Elites* pour l'environnement *BipedalWalker*.

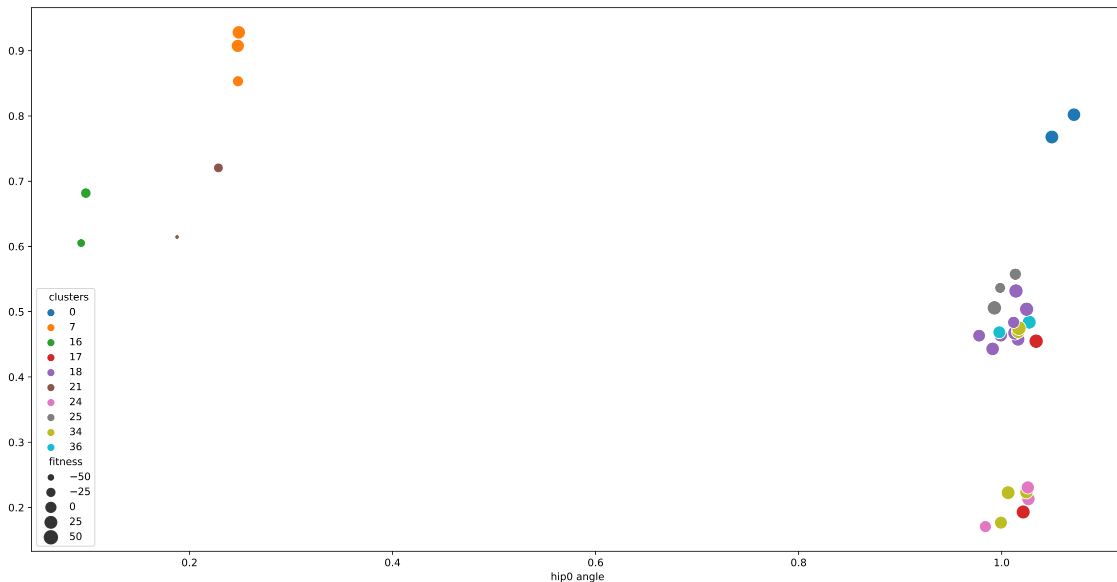



FIGURE 12 – En abscisse l'angle moyen de la première jambe, en ordonnée l'angle moyen de la seconde. Les features et fitness ont été évalués sur 50 simulations.

Tout d'abord, à partir d'un ensemble de point uniforme sur le domaine d'application du problème, les entrées-sorties des programmes sont comparées. Les programmes ayant les mêmes sorties qu'un autre sur l'ensemble de points sont considérés fonctionnellement équivalentes. Cette étape fait passer le nombre de programmes de la grille de 2483 à 441. Cela signifie, du fait de l'évaluation Monté-Carlo des programmes, qu'un même programme peut occuper plusieurs cellules. Ensuite, les programmes ont été regroupés en clusters à l'aide de la distance d'édition. Les programmes au sien d'un cluster n'ont au plus qu'une instruction d'écart. Du fait du grand nombre de clusters retournés, seuls les dix clusters avec la plus grande variance de comportement sont affichés sur la figure 12. Ainsi, on remarque que chaque cluster est concentré dans une même zone de l'espace de description du comportement. Cela nous indique en particulier que dans cette grille, il n'y a pas de programme *"modèle"* capable de générer plusieurs comportements. Ainsi, chaque programme semble dédié à un unique comportement.

# F Expérimentations Infructueuses

Cette annexe a pour but de présenter quelques expérimentations effectuées pendant le stage qui n'ont abouti à aucune amélioration.

**Optimisation des constantes** : Pour chacun des programmes et à chaque génération, des itérations d'optimisation locale sont appliquées sur les poids du programme. L'optimisation est effectuée via *evolutionnay strategies* [39]. Malheureusement utiliser ce processus est bien trop gourmand en simulation ce qui augmente significativement le temps calcul par génération et ne produit pas de résultats satisfaisants. Éventuellement, une approche utilisant de l'*importance sampling* tel que *PPO* [27] pourrait rendre l'optimisation plus viable en réduisant le nombre d'appels au simulateur.

**Modularité :** La modularité permet à la programmation génétique de réutiliser du code au sien d'un programme. Une méthode classique pour y parvenir repose sur les *Automatically Defined Function* (ADF) [40] qui définissent une hiérarchie de fonctions à évoluer. Cette méthode, appliquée à notre problème, ne donne pas de résultats intéressants. D'une part, les performances des politiques ne sont pas améliorées et d'autre part, elles sont plus difficiles à interpréter. En effet, les modules trouvés ne semblent pas avoir de sens particulier et ont tendance à s'appeler entre eux produisant un enchevêtrement de fonctions particulièrement illisible.

***Value function :*** Nous avons essayé de reproduire les résultats de *Kubalík et al.* [14] concernant l'utilisation de la régression symbolique pour décrire la *Value function*. Sans code de référence et malgré les détails donnés par les auteurs que nous avons contactés, nous n'avons jamais réussi à obtenir d'expression viable de la *Value function* via régression symbolique.

# Références

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015. 1

[2] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, *et al.*, “Explainable artificial intelligence (XAI) : concepts, taxonomies, opportunities and challenges toward responsible ai,” *Information Fusion*, vol. 58, pp. 82–115, 2020. 1

[3] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning,” *arXiv :1702.08608*, 2017. 2

[4] J. R. Koza *et al.*, *Genetic programming II*, vol. 17. MIT press Cambridge, 1994. 2, 5

[5] M. F. Brameier and W. Banzhaf, *Linear genetic programming.* Springer Science & Business Media, 2007. 2, 5

[6] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-based batch mode reinforcement learning,” *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005. 2

[7] G. Liu, O. Schulte, W. Zhu, and Q. Li, “Toward interpretable deep reinforcement learning with linear model u-trees,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 414–429, 2018. 2

[8] O. Bastani, Y. Pu, and A. Solar-Lezama, “Verifiable reinforcement learning via policy extraction,” *arXiv :1805.08328*, 2018. 2

[9] Y. Coppens, K. Efthymiadis, T. Lenaerts, A. Nowé, T. Miller, R. Weber, and D. Magazzeni, “Distilling deep reinforcement learning policies in soft decision trees,” in *Workshop on Explainable Artificial Intelligence*, pp. 1–6, 2019. 2

[10] A. M. Roth, N. Topin, P. Jamshidi, and M. Veloso, “Conservative q-improvement : Reinforcement learning for an interpretable decision-tree policy,” *arXiv :1907.01180*, 2019. 2

[11] F. Maes, R. Fonteneau, L. Wehenkel, and D. Ernst, “Policy search in a space of simple closed-form formulas : Towards interpretability of reinforcement learning,” in *International Conference on Discovery Science*, pp. 37–51, 2012. 2, 3

[12] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, “Programmatically interpretable reinforcement learning,” in *International Conference on Machine Learning*, pp. 5045–5054, 2018. 2, 3

[13] V. Liventsev, A. Härmä, and M. Petković, “Neurogenetic programming framework for explainable reinforcement learning,” *arXiv :2102.04231*, 2021. 2, 3

[14] J. Kubalík, J. Žegklitz, E. Derner, and R. Babuška, “Symbolic regression methods for reinforcement learning,” *arXiv :1903.09688*, 2019. 2, 3, 23

[15] D. Hein, S. Udluft, and T. A. Runkler, “Interpretable policies for reinforcement learning by genetic programming,” *Engineering Applications of Artificial Intelligence*, vol. 76, pp. 158–169, 2018. 2, 3, 10

[16] D. G. Wilson, S. Cussat-Blanc, H. Luga, and J. F. Miller, “Evolving simple programs for playing atari games,” in *Genetic and Evolutionary Computation Conference*, pp. 229–236, 2018. 2, 3

[17] H. Zhang, A. Zhou, and X. Lin, "Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis," *Complex & Intelligent Systems*, vol. 6, no. 3, pp. 741–753, 2020. 2, 3, 10

[18] M. Landajuela, B. K. Petersen, S. Kim, C. P. Santiago, R. Glatt, N. Mundhenk, J. F. Pettit, and D. Faissol, "Discovering symbolic policies with deep reinforcement learning," in *International Conference on Machine Learning*, pp. 5979–5989, 2021. 2, 3, 10

[19] J. R. Koza, M. A. Keane, J. Yu, F. H. Bennett, and W. Mydlowec, "Automatic creation of human-competitive programs and controllers by means of genetic programming," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1, pp. 121–164, 2000. 3

[20] D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T. A. Runkler, and V. Sterzing, "A benchmark environment motivated by industrial control problems," in *IEEE Symposium Series on Computational Intelligence*, pp. 1–8, 2017. 3

[21] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm : Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002. 5

[22] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP : Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012. 6

[23] L. Cazenille, "Qdpy : A python framework for quality-diversity." `https://gitlab.com/leo.cazenille/qdpy`, 2018. 6

[24] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3." `https://github.com/DLR-RM/stable-baselines3`, 2019. 6

[25] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. 6

[26] B. Ellenberger, "Pybullet gymperium." `https://github.com/benelot/pybullet-gym`, 2018–2019. 6

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv :1707.06347*, 2017. 7, 23

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, pp. 1861–1870, 2018. 7, 13

[29] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, pp. 1928–1937, 2016. 7, 13

[30] A. Raffin, "Rl baselines3 zoo." `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020. 7, 13

[31] T. Cazenave, "Nested monte-carlo search," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009. 7

[32] T. Cazenave, "Nested monte-carlo expression discovery," in *ECAI*, pp. 1057–1058, 2010. 7

[33] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011. 10

[34] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv :1504.04909*, 2015. 12

[35] M. Flageat and A. Cully, "Fast and stable map-elites in noisy domains using deep grids," in *Artificial Life Conference*, pp. 273–282, 2020. 12

[36] E. Real, C. Liang, D. So, and Q. Le, "Automl-zero : Evolving machine learning algorithms from scratch," in *International Conference on Machine Learning*, pp. 8007–8019, 2020. 14

[37] A. Gaier, A. Asteroth, and J.-B. Mouret, "Data-efficient exploration, optimization, and modeling of diverse designs through surrogate-assisted illumination," in *Genetic and Evolutionary Computation Conference*, pp. 99–106, 2017. 14

[38] S. Kelly, R. J. Smith, and M. I. Heywood, "Emergent policy discovery for visual reinforcement learning through tangled program graphs : A tutorial," *Genetic programming theory and practice XVI*, pp. 37–57, 2019. 15

[39] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arxiv :1703.03864*, vol. abs/, 2017. 23

[40] J. R. Koza, "Hierarchical automatic function definition in genetic programming," in *Foundations of Genetic Algorithms*, vol. 2, pp. 297–318, 1993. 23