L'optimisation multi-tâche et ses applications à la robotique : d'abord résoudre, ensuite généraliser

Timothee Anne

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

L'optimisation multi-tâche et ses applications à la robotique : d'abord résoudre, ensuite généraliser

THÈSE

présentée et soutenue publiquement le 6 juin 2024

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Timothée Anne

Composition du jury

Président :	Bernardetta Addis	LORIA - Université de Lorraine
Rapporteurs :	Clément Moulin-Frier	Centre Inria de l'université de Bordeaux
	Marc Schoenauer	Centre Inria de Saclay
Examinateur :	Amy Hoover	New Jersey Institute of Technology
Directeur de thèse :	Jean-Baptiste Mouret	Centre Inria de l'Université de Lorraine

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Résumé

Doter des agents artificiels, tels que des robots, d'une capacité à apprendre à réaliser des tâches complexes et à s'adapter est une quête centrale de la recherche en intelligence artificielle. L'apprentissage par renforcement profond en est aujourd'hui une des méthodes privilégiées, mais n'est ni toujours simple à mettre en œuvre, ni toujours la plus performante.

Dans cette thèse, nous étudions un autre concept d'apprentissage de politique qui se divise en deux étapes : une étape de résolution d'un ensemble de sous-problèmes puis une étape de généralisation. Plus formellement, la première étape reformule le problème général comme un problème multi-tâche permettant d'obtenir un jeu de données de solutions. La seconde étape utilise de l'apprentissage supervisé sur ce jeu de données pour entraîner une politique générale.

Nous évaluons d'abord la viabilité de ce concept à un problème d'apprentissage de réflexes d'évitement de chute avec un robot humanoïde réel. Non seulement il permet d'apprendre des comportements en simulation qui permettent d'éviter la chute dans plus de 75% des cas, mais ces comportements sont assez robustes pour fonctionner sur le robot réel.

Nous développons ensuite un algorithme de qualité-diversité multi-tâche, *Multi-Task Multi-Behavior MAP-Elites*, pour améliorer l'efficacité d'échantillonnage de la première étape de résolution. Nous illustrons cet algorithme sur le même problème d'apprentissage de réflexes d'évitement de chute d'un robot humanoïde et pour généraliser à des environnements plus réalistes.

Nous proposons enfin de passer d'une étape de résolution discrète à une résolution continue. Pour ce faire, nous reformulons le problème d'optimisation multi-tâche boîte noire comme un problème d'optimisation paramétrique et proposons une méthode pour le résoudre : *Parametric-Task MAP-Elites*. *Parametric-Task MAP-Elites* résout une nouvelle tâche à chaque itération, recouvrant asymptotiquement l'espace des tâches. Après avoir consommé son budget d'évaluations, *Parametric-Task MAP-Elites* distille les solutions trouvées dans une politique pour généraliser à l'ensemble de l'espace continu.

L'optimisation multi-tâche est une méthode sous-exploitée qui montre, dans cette thèse, son aptitude à permettre de résoudre certains problèmes de robotique de façon plus simple à mettre en œuvre et plus performante que l'apprentissage par renforcement profond.

Mots-clés: Intelligence artificielle, Robotique évolutionniste, Apprentissage par renforcement, Optimisation Multi-Tâche, Qualité-Diversité.

Abstract

Granting artificial agents, such as robots, the capability to learn how to solve complex tasks and adapt is a central quest in artificial intelligence research. Today, reinforcement learning is favored, but it is neither straightforward to implement nor consistently effective.

In this thesis, we explore an alternative policy learning concept divided into two stages : solving a set of sub-problems and generalization. More formally, the first stage reformulates the general problem into a multi-task problem to obtain a dataset of high-performing solutions. The second stage applies supervised learning to this dataset to train a general policy.

We first evaluate the viability of this concept in learning fall avoidance reflexes with a real humanoid robot. It enables learning behaviors in simulation that prevent falling in more than 75% of cases, and these behaviors are robust enough to function on the real robot.

We then develop a multi-task multi-behavior quality-diversity algorithm, Multi-Task Multi-Behavior MAP-Elites, to improve the sample efficiency of the first resolution stage. We demonstrate its application in fall avoidance reflex learning, where it performs better than a deep reinforcement learning algorithm and also enables generalization to more realistic environments.

Finally, we propose to go from a discrete resolution stage to a continuous resolution stage. To do so, we reformulate the multi-task black-box optimization problem as a parametric optimization problem and propose a method to solve it : Parametric-Task MAP-Elites. Parametric-Task MAP-Elites solves a new task at each iteration, asymptotically covering the task space. After consuming its evaluation budget, Parametric-Task MAP-Elites distills the solutions found into a policy to generalize to the entire continuous space.

Multi-task optimization is an underexploited method that has demonstrated in this thesis its ability to solve some robotics problems more straightforwardly and effectively than deep reinforcement learning.

Keywords: Machine Learning, Evolutionary Robotics, Reinforcement Learning, Multi-Task Optimization, Quality-Diversity.

Remerciements

Merci à Jean-Baptiste pour sa pédagogie, sa patience et ses encouragements.
Merci aux membres du bureau C127 et en particulier à AlXim, Assem, Jak et Teysä.
Merci à tous les joueurs et joueuses de la pause de midi.
Merci à Aude et Vlad pour les nombreuses parties de jeu le week-end.
Merci aux membres de l'équipe Larsen pour leur accueil et convivialité.
Merci au personnel de la cantine qui permet le partage d'un moment convivial.
Merci aux agents de l'accueil pour leurs sourires et salutations chaleureuses.
Merci à mes amis qui me soutiennent depuis de nombreuses années.
Merci à ma sœur et mes parents qui me soutiennent depuis le début.

"The most important step a man can take. It's not the first one, is it ?
It's the next one. Always the next step, Dalinar."
— Brandon Sanderson, Oathbringer
Parce que la thèse est faite de nombreux pas qu'il a fallu faire.
Parce que la thèse est juste un pas qui en entraînera un autre.

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Apprendre de nouveaux comportements est une compétence que l'humain développe dès le plus jeune âge. L'enfant apprend ainsi à marcher, parler et, plus globalement, à manipuler son corps et les objets qui l'entourent en interagissant avec son environnement (Gopnik et al., 1999). L'humain continue d'apprendre lors d'une longue phase dédiée à l'éducation. L'acquisition de compétences ne s'arrête pas là ; que ce soit par exemple pour maîtriser un nouveau sport ou un instrument de musique, la volonté d'en savoir plus et de faire plus guide l'humain dans son développement continu (Bransford et al., 2000).

Une quête centrale de l'intelligence artificielle est de parvenir à doter des agents artificiels de cette aptitude à apprendre. L'une des applications les plus directes est l'intelligence artificielle appliquée à la robotique. En effet, les robots sont devenus de plus en plus complexes, avec aujourd'hui une course au premier robot humanoïde généraliste [1] [2] [3]. Une preuve de cet engouement est une campagne de financement de plus de 600 millions d'euros en ce début d'année 2024 pour que la startup Figure AI conçoive un robot humanoïde s'appuyant fortement sur l'intelligence artificielle [4].

Pour mener à bien cette quête, de nombreux travaux en intelligence artificielle puisent une partie de leur inspiration du vivant (Sutton et Barto, 1998 ; Pfeifer et Bongard, 2006 ; Pfeifer, Lungarella et al., 2007 ; Iida et al., 2016). Nous pouvons distinguer trois grandes familles d'approches selon leur source d'inspiration.

Une première source d'inspiration est la cognition humaine et le développement chez l'enfant, qui ont inspiré les sciences cognitives, mêlant neuroscience, psychologie, intelligence artificielle, philosophie et anthropologie (Stillings et al., 1995). Les sciences cognitives et développementales développent des techniques s'inspirant de la manière dont l'humain, et surtout l'enfant, apprend et interagit avec le monde. Elles visent également à modéliser ces concepts pour mieux comprendre les mécanismes mis en jeu. L'étude de la curiosité et, en particulier, la motivation intrinsèque que développent les enfants pour apprendre à interagir avec le monde sont un exemple d'inspiration pour la robotique (Oudeyer et al., 2007). Une illustration de ces principes consiste à essayer de modéliser cette curiosité intrinsèque pour diriger l'exploration des comportements d'un robot (Baranes et al., 2013 ; Forestier et al., 2022).

1. https://spectrum.ieee.org/marc-raibert-boston-dynamics-instutute
2. https://spectrum.ieee.org/figure-humanoid-robot
3. https://spectrum.ieee.org/nvidia-gr00t-ros
4. https://spectrum.ieee.org/figure-robot-video

Une deuxième source d'inspiration est le principe d'évolution développé en biologie à partir de la théorie de l'évolution de Charles Darwin (Darwin, 1859), qui a inspiré le calcul évolutionniste[5] (Back et al., 1997) et la robotique évolutionniste (Nolfi et al., 2000 ; Doncieux, Mouret et al., 2011 ; Bongard, 2013 ; Doncieux, Bredeche et al., 2015a). Le calcul évolutionniste conçoit des heuristiques en s'appuyant le plus souvent sur des principes de populations d'individus qui évoluent en sélectionnant les individus les plus aptes pour effectuer des mutations et croisements afin de générer de nouveaux individus. Le calcul évolutionniste s'est développée à travers plusieurs approches distinctes. Initialement, la programmation évolutionnaire (Fogel et al., 1966) se concentre sur l'évolution des paramètres des programmes, utilisant à l'origine des machines à états finis pour prédire l'état de l'environnement. Puis, les algorithmes génétiques (Holland, 1992) tirent parti de l'encodage de l'information en séquences, souvent binaires, pour simuler les processus génétiques naturels. En parallèle, la programmation génétique (Koza et al., 1994) met l'accent sur l'évolution de programmes représentés le plus souvent par des arbres syntaxiques. Enfin, les stratégies d'évolution (Beyer et al., 2002) suivent un principe de sélection et de variation au sein d'une population d'individus, en se focalisant sur l'adaptation de ces variations pour améliorer la performance. La robotique évolutionniste s'inspire des mêmes idées pour les appliquer à la robotique en essayant de développer une intelligence incarnée (Pfeifer et Bongard, 2006). L'idée est de s'inspirer des êtres vivants qui sont le fruit d'une évolution conjointe d'un système physique et d'un contrôle cognitif. Pour la robotique, cela correspond à essayer de faire évoluer les robots dans un environnement en considérant les aspects de mécatronique, de traitement de l'information, de planifications et de contrôle comme un tout. Une des premières illustrations sont les travaux de Lipson et al. (2000) qui montrent qu'il est possible de faire évoluer en simulation des robots et de fabriquer les plus performants. Les robots correspondent à des articulations connectées par des barres rigides ou extensibles et actionnées à l'aide d'un réseau de neurones. L'algorithme évolue une population de tels robots (structure et connexion neuronale) avec pour *fitness*[6] (c'est-à-dire l'aptitude de l'individu à répondre au problème) la distance parcourue par les robots en simulation. Les robots les plus performants sont sélectionnés puis mutés en ajoutant, modifiant ou enlevant des articulations ou des barres pour former une nouvelle génération. Les trois robots les plus performants en simulation ont ensuite été fabriqués grâce à de l'impression 3D et ont montré dans le monde réel des comportements similaires à ceux en simulation.

Une troisième source d'inspiration est le principe de conditionnement opérant en psychologie comportementale, prenant son origine dans des expériences effectuées sur des animaux. Il a d'abord été étudié par Edward L. Thorndike avec des chats dans des labyrinthes (Thorndike, 1898), puis par B.F. Skinner avec des rats et des pigeons enfermés dans des boîtes de Skinner (Skinner, 1938). Cela a inspiré l'apprentissage par renforcement (Kaelbling et al., 1996), qui a lui-même été récemment développé grâce aux avancées en apprentissage profond pour devenir l'apprentissage par renforcement profond (Sutton et Barto, 1998 ; Mnih, Kavukcuoglu, Silver, Graves et al., 2013 ; Arulkumaran et al., 2017).

L'apprentissage par renforcement modélise généralement un comportement comme une succession de paires (action, état) qui rapportent une récompense mesurant la qualité de réalisation de la tâche (Sutton et Barto, 1998). L'idée principale est d'optimiser les actions prises à chaque pas de temps pour maximiser le gain de cette récompense tout au long du comportement. Cela n'est pas trivial, car il faut résoudre le problème d'attribution du crédit.

5. Dans ce manuscrit, nous avons décidé de traduire *evolutionary computation* par calcul évolutionniste.

6. Étant donné l'utilisation courante du terme anglais et la confusion que pourrait engendrer l'utilisation d'une traduction française, nous avons décidé d'employer le terme anglais dans l'ensemble de ce manuscrit de thèse.

En effet, une action au tout début de la séquence peut avoir des conséquences importantes à la fin, mais l'agent ne voit la récompense augmenter que bien plus tard. Par exemple, pour un problème d'exploration d'un labyrinthe, prendre une bifurcation plutôt qu'une autre peut conditionner le succès ou l'échec de l'agent.

Les premiers algorithmes d'apprentissage par renforcement utilisent des méthodes de programmation dynamique (Bellman, 1966) pour estimer la récompense moyenne que peut espérer l'agent en se plaçant dans un état. *Adaptive Heuristic Critic* (Barto et al., 1983) est l'un des premiers algorithmes qui apprend un critique sous forme d'une table de valeur des récompenses moyennes à long terme pour chaque état, utilisée pour optimiser l'action via différence temporelle. Le *Q-learning* (Watkins et al., 1992) développe ensuite cette idée pour apprendre une valeur de récompense moyenne pour chaque paire d'état et d'action. Il faudra attendre l'essor technique de l'apprentissage profond au début des années 2010 pour que la capacité d'approximation de fonction des réseaux de neurones profonds permette de généraliser l'apprentissage de la valeur de chaque paire d'action-état aux espaces de plus hautes dimensions et continues (LeCun, Bengio et al., 2015). Un exemple marquant a été l'apprentissage de jeux Atari à partir uniquement de pixels et non plus de caractéristiques prédéfinies (Mnih, Kavukcuoglu, Silver, Graves et al., 2013).

Aujourd'hui, l'apprentissage par renforcement profond est l'une des méthodes les plus répandues pour entraîner un agent artificiel à accomplir une tâche (Mnih, Kavukcuoglu, Silver, Rusu et al., 2015 ; Lillicrap et al., 2015a ; Arulkumaran et al., 2017). L'entreprise DeepMind, s'efforçant de démontrer les capacités de l'apprentissage profond, a créé des réalisations notables telles que AlphaGo (Silver, A. Huang et al., 2016), AlphaGo Zero (Silver, Schrittwieser et al., 2017), AlphaFold (Jumper et al., 2021), ou encore AlphaStar (Vinyals et al., 2019), qui a battu le champion du monde d'un jeu vidéo de stratégie en temps réel, *Starcraft*. Un exemple d'application en robotique concerne la locomotion de robots quadrupèdes (Tan, T. Zhang et al., 2018 ; Hwangbo et al., 2019), y compris dans des environnements en dehors du laboratoire (J. Lee et al., 2020 ; Miki et al., 2022). L'apprentissage par renforcement profond commence ainsi à remplacer les méthodes de contrôle classiques (Song et al., 2023).

L'apprentissage par renforcement profond présente de nombreux avantages :

— l'apprentissage se fait de bout en bout, c'est-à-dire que l'algorithme prend en entrée les données brutes telles que les capteurs de vision du robot et des informations sur l'actuation de chaque articulation, et contrôle en temps réel le robot pour réaliser la tâche en maximisant la récompense ;

— les réseaux de neurones permettent d'apprendre des politiques complexes qui résolvent des problèmes difficiles, surpassant souvent les performances humaines (par exemple AlphaStar) ;

— le processus de décision est distribué dans les poids du réseau de neurones, rendant le processus d'inférence beaucoup plus rapide que l'entraînement, ce qui permet de l'appliquer à du contrôle en temps réel (par exemple 50 Hz pour le contrôle d'un robot quadrupède (J. Lee et al., 2020)).

Néanmoins, bien que nous ne puissions nier les résultats impressionnants de l'apprentissage par renforcement profond, cette méthode est souvent difficile à appliquer à de nouveaux problèmes. En effet, l'apprentissage par renforcement profond doit résoudre simultanément trois problèmes : (1) le problème d'exploration des espaces d'états et d'actions et du lien entre les deux pour découvrir des comportements, (2) l'optimisation de ces comportements pour améliorer la qualité des solutions et (3) la généralisation de ces solutions pour fournir une politique globale de résolution du problème.

L'apprentissage par renforcement profond présente donc aussi de nombreux désavantages :

— l'entraînement est coûteux en interaction avec l'environnement, c'est-à-dire que l'algorithme n'a pas une bonne efficacité d'échantillonnage, ce qui est particulièrement fastidieux et coûteux dans des environnements où l'évaluation prend du temps (par exemple, l'entraînement d'AlphaStar est composé de 10 milliards d'observations soit plus d'une dizaine d'années de temps de jeu) et plus particulièrement dans le monde réel qui ne peut pas être accéléré et pour lequel la parallélisation coûte cher ;
— chaque entraînement nécessite, en plus, de recommencer cette collecte de données, ce qui se produit souvent, en particulier lors de la phase de paramétrage de la méthode ;
— l'exploration est souvent mise en second plan au profit d'une sur-exploitation des premières solutions trouvées, ce qui explique en partie la faible efficacité d'échantillonnage (ce qui peut être atténué en modifiant la fonction de récompense (Grzes, 2017), mais qui est souvent fastidieux, car cela nécessite d'itérer de nombreux entraînements) ;
— l'entraînement est aussi coûteux en temps d'ingénieur, c'est-à-dire que la majorité des algorithmes sont dotés d'un grand nombre d'hyperparamètres (par exemple, *Soft Actor-Critic* (Haarnoja et al., 2018) en possède plus d'une dizaine) pouvant avoir un fort impact sur les performances, de plus, des détails d'implémentation peuvent également avoir des impacts importants sur les performances, ce qui pose un problème de reproductibilité (Nagarajan et al., 2018 ; Henderson, 2018) ;
— la compréhension de l'origine d'un entraînement de mauvaise qualité est souvent difficile à cause de l'apprentissage de bout en bout, par exemple lorsque l'entraînement échoue, il est difficile de savoir si cela provient de la fonction de récompense, de la représentation, des hyperparamètres ou si le problème est impossible ;
— les comportements sont stockés dans les poids des réseaux de neurones de façon le plus souvent incompréhensible pour l'humain, ce qui rend difficile l'interprétation des choix pris par l'algorithme (Heuillet et al., 2021) ;
— l'impossibilité de contraindre directement les comportements, par exemple pour des raisons de sécurité, ce qui n'est que partiellement et indirectement corrigé par l'ajout de termes de pénalisation dans la récompense, sachant que l'algorithme n'est pas contraint de les respecter.

Dans cette thèse, nous nous intéressons plus particulièrement à chercher des méthodes alternatives qui ne sont pas de bout en bout. Cette caractéristique est, selon nous, une cause principale du besoin de recommencer la collecte de données à chaque apprentissage et la difficulté de déboguer un apprentissage qui se passe mal. De façon générale, nous pensons qu'un algorithme modulaire, c'est-à-dire composé de plusieurs sous-parties indépendantes, est plus simple et plus robuste.

En comparaison de la popularité de l'apprentissage par renforcement profond, peu de travaux explorent d'autres manières de trouver des politiques, potentiellement en exploitant aussi des réseaux de neurones profonds. De la même façon qu'un humain n'apprend pas uniquement grâce au principe de renforcement en interagissant avec l'environnement mais aussi par l'éducation (Bransford et al., 2000), plusieurs méthodes en robotique s'intéressent à l'apprentissage par démonstration ou imitation. Un exemple important est l'algorithme *Dataset Aggregation* (DAgger) (Ross et al., 2011) qui entraîne une politique à imiter les comportements d'un expert. *Guided Policy Search* (GPS) (Levine et Koltun, 2013) propose une méthode similaire mais remplace l'expert humain par des démonstrations et de l'optimisation de trajectoire optimale. *Go-Explore* (Ecoffet et al., 2021) s'affranchit de la nécessité d'avoir accès à un expert ou à des démonstrations et sépare l'apprentissage en une étape d'exploration pour trouver des comportements permettant d'explorer le plus d'états possible, puis une étape de généralisation via démonstration sur ces comportements. *Go-Explore* parvient, sur le jeu de problèmes

de référence Atari 2600 (Bellemare et al., 2013), à obtenir non seulement de meilleures performances que les méthodes d'apprentissage par renforcement profond, mais aussi que les records de score d'experts humains. De façon générale, ces méthodes divisent la résolution du problème en plusieurs étapes, ce qui permet d'utiliser, aux moments opportuns, des techniques ayant différents avantages et inconvénients.

Dans cette thèse, nous faisons l'hypothèse qu'un découpage viable est le suivant : (1) la résolution d'un ensemble de sous-problèmes du problème général et (2) la généralisation de ces solutions à l'ensemble du problème. L'étape (1) peut se voir comme un problème de pure optimisation sans nécessiter d'apprentissage et de généralisation. Alors que l'étape (2) se concentre uniquement sur la généralisation d'un problème déjà résolu et doit donc trouver comment interpoler et extrapoler les solutions trouvées lors de l'étape (1) à l'ensemble du problème.

Comparé à l'apprentissage par renforcement profond qui tente de résoudre l'intégralité du problème d'un seul coup, découper le problème en sous-problèmes lors de l'étape (1) permet de réduire la difficulté. D'une part, chaque sous-problème peut être plus simple à résoudre que de trouver une politique générale qui résout le problème global. D'autre part, chaque sous-problème peut être résolu avec des informations qui seront indisponibles au moment de l'inférence. Par exemple DAgger (Ross et al., 2011) utilise les démonstrations d'un expert et *Guided Policy Search* (Levine et Koltun, 2013) utilise un modèle de la dynamique. L'étape (2) peut ensuite se voir comme de l'apprentissage supervisé d'une politique générale.

À titre purement illustratif, prenons le problème d'apprendre à un robot humanoïde à ouvrir n'importe quelle porte. Pour un humain, ouvrir une nouvelle porte est une tâche facile qui ne prend rarement plus de deux essais. Ceci est dû au fait que nous avons appris que la majorité des portes possèdent une poignée et peuvent être poussées, tirées ou parfois coulissées.

Une méthode d'apprentissage par renforcement profond doit donc résoudre d'un seul coup plusieurs aspects tels que la détection de porte, la classification du type de porte, la locomotion pour se positionner autour de la porte, le contrôle de la manipulation pour ouvrir la porte, suivi d'une nouvelle phase de locomotion pour passer la porte, sachant que certaines portes nécessitent parfois d'être maintenues ouvertes, impliquant un mouvement plus dynamique. Résoudre globalement tous ces aspects du problème pour tous les types de portes semble être une tâche difficile. En revanche, en divisant ce problème en deux étapes : (1) une étape de résolution d'une subdivision du problème en un ensemble de sous-problèmes, chacun plus simple que le problème général, et (2) une étape de généralisation des solutions trouvées, le problème semble plus abordable.

Pour résoudre l'étape (1), une méthode potentielle consisterait à instancier le problème pour de nombreux cas particuliers et traiter chacun de ces sous-problèmes indépendamment. Si, pour chaque sous-problème, l'algorithme a accès à un modèle du robot et de l'environnement, une méthode de planification peut aussi être envisagée (LaValle, 2006). Si, au contraire, l'algorithme n'a pas accès à un modèle analytique, des méthodes d'optimisation boîte-noire peuvent être utilisées (par exemple, CMA-ES (Hansen et al., 2003)). Il y a aussi un potentiel d'utiliser une méthode plus efficace que de résoudre chaque sous-problème individuellement. En effet, des sous-problèmes similaires ont probablement des solutions proches. Par exemple, un humain qui sait ouvrir une porte en tournant la poignée et en tirant, sait certainement ouvrir une porte qu'il n'a jamais vue si celle-ci possède aussi une poignée similaire.

La plus grande partie des méthodes d'optimisation ne s'intéresse qu'à optimiser une unique fonction à la fois, seules quelques méthodes s'intéressent à en optimiser plusieurs simultanément. Nous pouvons ainsi citer : *Multi-Task MAP-Elites* (Mouret et Maguire, 2020), une méthode d'optimisation multi-tâche en robotique évolutionniste ; les méthodes d'évolution

multifactorielle en calcul évolutionniste (Gupta, Y.-S. Ong et al., 2016) ; ou encore la programmation paramétrique (Pappas et al., 2021) en optimisation classique. Ce type d'optimisation n'ayant pas de terme communément utilisé, nous l'appelons dans cette thèse l'optimisation multi-tâche. Ce problème correspond à résoudre un ensemble de tâches, chacune correspondant à une fonction différente à optimiser.

L'idée sous-jacente derrière l'optimisation multi-tâche est d'exploiter la similarité entre les différentes fonctions à optimiser pour accélérer la résolution. L'ensemble des solutions que nous cherchons peut être vu comme une population d'individus en compétition pour dominer un ensemble de niches écologiques correspondant à chaque fonction. À partir de cette observation, il est par exemple possible d'utiliser les principes d'évolution employés par la robotique évolutionniste pour optimiser l'ensemble des solutions (Mouret et Maguire, 2020). *Multi-Task MAP-Elites* (Mouret et Maguire, 2020) est l'un des rares algorithmes qui applique ces principes pour résoudre des problèmes d'optimisation multi-tâche composés d'un grand nombre de tâches (c'est-à-dire plusieurs milliers).

Dans cette thèse, nous supposons qu'une méthode d'optimisation multi-tâche est une candidate appropriée pour l'étape (1). D'une part, la résolution multi-tâche accélère le processus en exploitant les similarités entre les différentes tâches. D'autre part, l'optimisation est une méthode plus générale que, par exemple, la planification en robotique, ce qui lui permet de s'appliquer à un champ plus large de problèmes.

Si nous revenons à notre exemple d'ouverture de porte, l'étape (1) pourrait consister à d'abord sélectionner un grand nombre de situations différentes (par exemple, le type de porte et de poignée, le placement dans l'environnement et l'encombrement du couloir), chacune de ces situations correspondant à un sous-problème particulier du problème global. Ensuite, nous pourrions utiliser un algorithme qui a connaissance de ces descriptions de situation pour optimiser une politique dédiée à l'ouverture de cette porte dans cette situation particulière. Ce type d'apprentissage s'appelle l'apprentissage privilégié (Vapnik et al., 2009). Il est, par exemple, utilisé pour d'abord apprendre un contrôleur de marche d'un robot quadrupède en simulation qui est ensuite utilisé pour obtenir un contrôleur robuste pour le monde réel ayant accès à beaucoup moins d'information (J. Lee et al., 2020 ; Miki et al., 2022). Un exemple dans notre problème d'ouverture de porte pourrait être d'utiliser une méthode de planification qui a connaissance du modèle de la porte (Chitta et al., 2010) tout en amorçant la planification avec une solution d'un autre sous-problème déjà résolue.

Résoudre l'étape (2) correspond à généraliser en extrapolant d'un jeu de données, ce qui relève de l'apprentissage supervisé (Cunningham et al., 2008). Quelques-unes de ces méthodes les plus connues sont la régression linéaire, la machine à vecteurs de support, les arbres de décision, l'algorithme des k plus proches voisins, ou encore les réseaux de neurones. Ces derniers sont particulièrement intéressants grâce à leur capacité à théoriquement pouvoir approcher n'importe quelle fonction (Hornik et al., 1989 ; Cybenko, 1989). Il a fallu attendre le début des années 2010 pour que les réseaux de neurones profonds deviennent dominant en vision avec le succès d'*AlexNet* (Krizhevsky et al., 2012) dans la classification d'images en haute définition, qui a surpassé les méthodes concurrentes lors de la compétition ILSVRC-2012 (Russakovsky et al., 2015) sur le jeu de données ImageNet. Nous pouvons aussi remarquer que les grands modèles de langages (LLMs) (T. Brown et al., 2020), qui tendent vers une intelligence générale, est une méthode d'apprentissage auto-supervisé qui exploite des méthodes d'apprentissage supervisé.

Pour conclure sur notre exemple d'apprentissage de comportement pour qu'un robot puisse ouvrir n'importe quelle porte, l'étape (2) pourrait consister à utiliser le jeu de données généré lors de l'étape (1) pour entraîner de façon privilégiée un réseau de neurones convolutionnel (Gu

et al., 2018). De cette façon, le robot pourrait être capable de, non seulement réaliser le comportement d'ouverture de porte avec uniquement des données visuelles, mais aussi de généraliser à de nouvelles portes.

1.2 Problématique et objectif

La thèse présentée dans ce manuscrit s'intitule "L'optimisation multi-tâche et ses applications à la robotique : d'abord résoudre, ensuite généraliser". Ayant réalisé cette thèse dans un centre de recherche disposant de robots humanoïdes, j'ai trouvé pertinent d'appliquer l'apprentissage de comportements sur ces machines complexes faisant encore aujourd'hui l'objet de recherches à la frontière de la robotique, du contrôle et de l'apprentissage.

Les robots humanoïdes sont des machines polyvalentes (Goswami et al., 2018). Ils ont la capacité de saisir, pousser, tirer, tenir et atteindre des endroits aussi bien bas que haut. De plus, ils sont capables de marcher, monter et descendre des escaliers, courir et ramper dans un tunnel. Grâce à leur posture bipède, ils peuvent naviguer dans des endroits étroits et, de manière générale, dans tous les environnements où un humain peut se déplacer.

Cette posture bipède rend aussi les robots humanoïdes instables, ils ont ainsi tendance à tomber pour la moindre erreur (Atkeson et al., 2018). En raison de leur complexité et de leur fragilité, la chute d'un robot humanoïde a de fortes chances d'entraîner des dégâts supplémentaires, signant souvent la fin de la mission pour le robot. En comparaison, chez l'humain, en cas de perte d'équilibre ou même d'une douleur apparaissant dans les jambes, celui-ci cherche par réflexe à assurer son équilibre en s'appuyant sur tout ce qu'il peut. Malheureusement, les robots humanoïdes ne sont pas encore dotés de tels réflexes. En effet, cela suppose une dynamique multi-contact difficile à planifier rapidement par les méthodes classiques utilisées pour contrôler les robots humanoïdes (Kheddar et al., 2019 ; Polverini et al., 2021).

Un réflexe doit être un comportement rapide (c'est-à-dire largement inférieur à 1 s) déclenché "instinctivement" en cas de détection d'un problème quelconque qui a de fortes chances d'engendrer une chute (par exemple un capteur qui ne répond plus dans la jambe du robot). Le but de ce comportement est de prévenir une chute éventuelle en retrouvant une position d'équilibre grâce à un ou plusieurs contacts supplémentaires sur l'environnement proche.

Ce problème amène à la question suivante :

Question 1.1

Est-il possible d'apprendre un réflexe à un robot humanoïde en découpant l'apprentissage en une première étape de résolution et une seconde étape de généralisation ?

Le chapitre 3 présente une méthode pour apprendre un réflexe à un robot humanoïde en utilisant un seul bras. La méthode découpe l'apprentissage en deux étapes : (1) la résolution d'un ensemble d'instances du problème de façon indépendante et archivage de chaque solution et (2) la généralisation de ces solutions à l'ensemble du problème en utilisant de l'apprentissage supervisé. Les réflexes sont appris avec une simulation du robot et sont évalués quantitativement en simulation et qualitativement sur le robot réel, nous permettant de répondre positivement à cette première question. Le chapitre 4 étudie l'extension de la méthode précédente aux deux bras, ce qui nécessite le développement d'une nouvelle méthode d'optimisation multi-tâche provenant de la robotique évolutionniste et évalue sa généralisation à des environnements plus réalistes.

La conclusion de ces deux chapitres est que l'utilisation d'une méthode d'optimisation multi-tâche pour l'étape (1) permet d'améliorer les performances globales de la méthode et qu'un simple classifieur appris par apprentissage supervisé est une bonne méthode pour l'étape (2). En effet, les méthodes d'apprentissage supervisé sont matures et performantes, la plus grande difficulté du problème étant de trouver comment le reformuler pour pouvoir les appliquer. Ceci nous amène à la question suivante :

Question 1.2

Est-il possible d'améliorer la première phase de résolution en proposant un algorithme d'optimisation multi-tâche boîte noire qui résout plus de tâches ?

Le chapitre 5 apporte des éléments de réponse à cette seconde question en s'éloignant de l'application sur un robot humanoïde. Il formule notre apprentissage en deux étapes comme un problème d'optimisation paramétrique boîte-noire et propose un nouvel algorithme qui permet (1) de résoudre une nouvelle instance du problème à chaque itération et (2) de distiller ces solutions dans un réseau de neurones pour effectivement proposer une solution pour chaque possible instance du problème.

1.3 Contributions et liste des publications

Ce manuscrit de thèse fait l'objet de 3 publications. Deux ont été acceptées par un jury de pairs et publiées dans des journaux ou conférences, et l'une dont les résultats nous semblais moins impactant a été soumise et présentée en tant que poster lors d'une conférence. La première publication présente D-Reflex, une méthode pour permettre à un robot humanoïde de s'appuyer sur un mur avec une main en cas de dommage pour éviter la chute. La seconde publication présente *Multi-Task Multi-Behavior MAP-Elites*, une variante de MAP-Elites (Mouret et Clune, 2015) et *Multi-Task MAP-Elites* (Mouret et Maguire, 2020) pour résoudre un grand nombre de tâches tout en obtenant pour chaque tâche le plus de solutions diverses. La troisième publication présente *Parametric-Task MAP-Elites*, un algorithme d'optimisation multi-tâche continue et boîte noire qui s'inspire de *Multi-Task MAP-Elites* (Mouret et Maguire, 2020).

Voici la liste complète des publications abordées durant cette thèse, accompagnées de leur résumé.

— **Anne, Timothée**, Dalin, Eloïse, Bergonzani, Ivan, Ivaldi, Serena et Mouret, Jean-Baptiste (2022). « First Do Not Fall : Learning to Exploit a Wall With a Damaged HumanoidRobot ». IEEE Robotics Autom. Lett. 7.4, p. 9028-9035. https://doi.org/10.1109/LRA.2022.3188884.

Résumé : Les robots humanoïdes pourraient remplacer les humains dans des situations dangereuses, mais ces situations sont tout aussi dangereuses pour eux, ce qui signifie qu'ils ont une forte probabilité de subir des dommages et de chuter. Nous faisons l'hypothèse que les robots humanoïdes seront le plus souvent déployés dans des bâtiments, ce qui implique une forte probabilité qu'ils soient proches d'un mur. Pour éviter la chute, ils peuvent donc s'appuyer sur le mur le plus proche, comme le ferait un humain, à condition qu'ils soient capables de choisir en quelques millisecondes où mettre la ou les main(s). Cet article introduit une méthode, que nous appelons D-Reflex, qui entraîne un réseau de neurones pour permettre de choisir cette position de contact sur le mur étant donné la distance et l'orientation du mur par rapport au

robot, ainsi que la posture du robot. Cette position de contact est ensuite utilisée par un contrôleur corps complet pour permettre au robot d'atteindre une posture stable. Nous montrons que D-Reflex permet au robot simulé TALOS (1.75 m, 100 kg, 30 degrés de liberté) d'éviter plus de 75% des chutes évitables et peut être déployé sur le robot réel.

— **Anne, Timothée** et Mouret, Jean-Baptiste (2023). « Multi-Task Multi-Behavior MAP-Elites ». POSTER. Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023. Sous la dir. de Sara Silva et Paquete. ACM, p. 111-114. https://doi.org/10.1145/3583133.3590730.

Résumé : Nous présentons *Multi-Task Multi-Behavior MAP-Elites*, une variante de MAP-Elites qui trouve un grand nombre de solutions de qualité pour un large ensemble de tâches (problèmes d'optimisation d'une famille donnée). Il combine le MAP-Elites original pour la recherche de diversité et *Multi-Task MAP-Elites* pour profiter des similitudes entre les tâches. Il est plus performant que trois méthodes de référence sur un ensemble de tâches liées à la récupération après des dommages sur un robot humanoïde, résolvant plus de tâches et trouvant deux fois plus de solutions par tâche résolue.

— **Anne, Timothée** et Mouret, Jean-Baptiste (2024). « Parametric-Task MAP-Elites ». Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2024, Companion Volume, Melbourne, Australia, July 14-18, 2024. ACM[7]. https://doi.org/10.1145/3638529.3653993

Résumé : Optimiser un ensemble de fonctions simultanément en exploitant leur similarité s'appelle l'optimisation multi-tâche. Les algorithmes d'optimisation multi-tâche boîte noire actuels ne résolvent qu'un nombre fini de tâches, même lorsque les tâches proviennent d'un espace continu. Dans ce papier, nous introduisons *Parametric-Task MAP-Elites* (PT-ME), un nouvel algorithme boîte noire pour résoudre des problèmes d'optimisation multi-tâche continue. Cet algorithme (1) résout une nouvelle tâche à chaque itération, recouvrant ainsi l'espace continu, et (2) exploite un nouvel opérateur de variation basé sur une régression linéaire locale. Le jeu de données de solutions résultant rend possible la création d'une fonction qui lie n'importe quelle paramétrisation de tâche à sa solution optimale. Nous montrons, sur deux problèmes jouets de tâches paramétriques et un problème de robotique plus réaliste et difficile en simulation, que PT-ME surpasse toutes les méthodes de référence, y compris l'algorithme d'apprentissage par renforcement profond PPO.

J'ai aussi travaillé sur deux autres publications acceptées par un jury de pairs et publiées dans une conférence :

— Rituraj, Kaushik, **Anne, Timothée**, et Mouret, Jean-Baptiste (2020). "Fast online adaptation in robotics through meta-learning embeddings of simulated priors." 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020. https://doi.org/10.1109/IROS45743.2020.9341462

Résumé : Les algorithmes de méta-apprentissage peuvent accélérer les algorithmes d'apprentissage par renforcement basé sur un modèle (MBRL) en trouvant un ensemble initial de paramètres pour le modèle dynamique, de sorte que le modèle puisse être entraîné pour correspondre à la dynamique réelle du système avec seulement quelques points de données. Cependant, dans le monde réel, un robot peut rencontrer n'im-

7. Accepté mais pas encore publié

porte quelle situation, depuis des pannes de moteur jusqu'à se retrouver sur un terrain rocheux où la dynamique du robot peut être considérablement différente les unes des autres. Dans cet article, nous montrons tout d'abord que lorsque les situations de méta-entraînement (les situations antérieures) ont des dynamiques si diverses, l'utilisation d'un seul ensemble de paramètres de méta-entraînement comme point de départ nécessite toujours un grand nombre d'observations du système réel pour apprendre. un modèle utile de la dynamique. Deuxièmement, nous proposons un algorithme appelé FAMLE qui atténue cette limitation en méta-entraînant plusieurs points de départ initiaux (c'est-à-dire des paramètres initiaux) pour entraîner le modèle et permet au robot de sélectionner le point de départ le plus approprié pour adapter le modèle à la situation actuelle avec seulement quelques étapes de gradient. Nous comparons FAMLE au MBRL, au MBRL avec un modèle méta-entraîné avec MAML et l'algorithme de recherche de politique sans modèle PPO pour diverses tâches robotiques simulées et réelles, et montrons que FAMLE permet aux robots de s'adapter à de nouveaux dommages en moins de temps que les méthodes de référence.

— **Anne, Timothée**, Wilkinson, Jack, and Li, Zhibin (2021). "Meta-learning for fast adaptive locomotion with uncertainties in environments and robot dynamics." 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2021. https://doi.org/10.1109/IROS51168.2021.9635840

Résumé : Ce travail utilise du méta-apprentissage pour développer des politiques de contrôle afin de réaliser une adaptation en ligne rapide à différentes conditions en changement constant et permettant d'obtenir une locomotion robuste. La méthode proposée met constamment à jour le modèle d'interaction, échantillonne des séquences réalisables d'actions de trajectoires état-action estimées, puis applique les actions optimales pour maximiser la récompense. Pour réaliser une adaptation du modèle en ligne, notre méthode apprend différents vecteurs latents de chaque condition d'entraînement, qui sont sélectionnés en ligne en fonction des données nouvellement collectées à partir des 10 derniers échantillons en 0,2 s. Notre travail utilise un espace d'état et des fonctions de récompense appropriés, et optimise des actions réalisables à la manière d'une commande prédictive qui sont échantillonnées directement dans l'espace articulaire en respectant des contraintes, ce qui ne nécessite donc aucun pré-apprentissage ou paramétrisation d'une marche. Nous avons en outre démontré la capacité du robot à détecter des changements inattendus pendant l'interaction et à adapter la politique de contrôle en moins de 0,2 s. La validation extensive sur le robot SpotMicro dans une simulation physique montre des compétences de locomotion adaptatives et robustes sous des conditions de friction du sol changeantes, des poussées externes, et différentes dynamiques robotiques incluant des pannes de moteur et l'amputation complète d'une jambe.

Ces deux travaux s'intéressent à l'utilisation du méta-apprentissage par renforcement pour l'adaptation de la locomotion de robots quadrupèdes. Bien que cela ait pu entrer dans une problématique d'apprentissage pour l'adaptation d'agents artificiels, nous avons décidé de ne pas les inclure. Cela permet de concentrer la problématique de cette thèse sur le concept d'apprentissage en deux étapes : "résoudre puis généraliser", qui nous semble aujourd'hui plus intéressant.

TABLE 1.1 – Sujets abordés dans chaque publication. ∗ signifie que l'information se trouve dans le complément de la publication (présent dans cette thèse), mais n'est pas incluse dans la publication.

	First do not fall (Chapitre 3)	**Multi-Task Multi-Behavior MAP-Elites (Chapitre 4)**	**Parametric-Task MAP-Elites (Chapitre 5)**
Multi-Tâche	non	oui	oui
D'abord, résoudre	oui	oui	oui
Ensuite, généraliser	oui	oui*	oui
Robotique évolutionniste	non	oui	oui
Comparaison à l'apprentissage par renforcement profond	non	oui*	oui
Robotique	oui (robot réel)	oui (simulation)	un peu
Qualité-Diversité	oui (exhaustif)	oui (MAP-Elites)	non

Plan

Le chapitre 2 propose d'abord une revue générale de l'apprentissage par renforcement profond et de la robotique évolutionniste avec un accent particulier sur la qualité-diversité et l'algorithme MAP-Elites (Mouret et Clune, 2015) sur lequel s'appuient les chapitres 4 et 5. Il présente ensuite différentes méthodes d'apprentissage qui explorent des alternatives à l'apprentissage par renforcement profond.

Les trois chapitres suivants présentent les contributions de cette thèse (le tableau 1.1 détaille les différents aspects abordés). Le chapitre 3 présente les travaux de Anne, Dalin et al. (2022), le chapitre 4 présente les travaux de Anne et Mouret (2023) et le chapitre 5 présente les travaux de Anne et Mouret (2024).

Enfin, le chapitre 6 propose une discussion des approches abordées et le chapitre 7 conclut cette thèse.

Chapitre 2

Revue de la littérature

Sommaire

2.1 Introduction

La principale problématique de cette thèse est l'étude d'une méthode d'apprentissage de politique qui propose une alternative à l'apprentissage par renforcement profond. La section 2.2 définit d'abord quelques termes clés. La section 2.3 propose un tour d'horizon de l'apprentissage par renforcement profond. La section 2.4 réalise une revue des méthodes de stratégies d'évolution pour l'apprentissage par renforcement profond. La section 2.5 présente le domaine de recherche de la robotique évolutionniste. Enfin, la section 2.6 met en lumière des méthodes d'apprentissage de politiques aujourd'hui sous-exploitées par rapport à l'apprentissage par renforcement profond.

2.2 Définitions

Cette section clarifie des termes utilisés tout au long de cette thèse qui, d'une communauté à l'autre, ne représentent pas toujours la même chose.

- **Action :** Une action est ce que l'agent peut choisir de faire pour interagir avec l'environnement. Pour un robot, cela peut être, par exemple, une commande en position ou en couple, mais peut aussi être une commande de plus haut niveau comme se déplacer à gauche, à droite, en avant, ou en arrière.
- **Agent :** Un agent est une entité se trouvant dans un environnement qui peut percevoir celui-ci soit partiellement à partir d'une observation o, soit globalement à partir de son état s, et qui peut interagir en réalisant une action a. Il peut par exemple s'agir d'un robot dans le monde réel ou en simulation, mais aussi d'un personnage dans un jeu vidéo.
- **Comportement :** Un comportement est un terme englobant représentant une description de ce que fait l'agent après avoir effectué une action (ou une série d'actions). L'agent ne choisit pas directement quel comportement il va effectuer, ce qui pose le problème d'apprendre quelle action entraîne quel comportement. Dans le cas de la locomotion d'un robot, un comportement peut être, par exemple, la vitesse de déplacement engendrée par une commande ou le taux de contacts de chaque membre avec le sol.
- **Politique :** Une politique est ce qui permet à l'agent de savoir quelle action faire pour un état donné. Il peut par exemple s'agir d'une séquence d'actions prédéfinies, ou d'une fonction ou d'un algorithme qui prend en entrée l'état de l'agent et de l'environnement et qui renvoie une action. En robotique classique, un exemple est un contrôleur d'optimisation quadratique qui optimise une commande à chaque pas de temps.
- **Tâche :** Une tâche est définie comme un problème à résoudre par l'agent dans un environnement donné et est caractérisée par une fonction de récompense ou de *fitness* qui mesure la résolution du problème. En robotique, il peut s'agir, par exemple, d'arriver à se déplacer à une certaine vitesse ou de saisir un objet et de le déposer près d'une cible.

2.3 L'apprentissage par renforcement profond

Pour modéliser l'interaction d'un agent dans un environnement, nous pouvons distinguer deux approches courantes : soit à l'aide d'une politique séquentielle, soit à l'aide d'une politique globale (Deisenroth, Neumann et al., 2013 ; Chatzilygeroudis, Vassiliades et al., 2019 ; Kroemer et al., 2021). La politique séquentielle découpe le comportement en une séquence d'actions qui entraîne une séquence d'états de l'environnement. Chacune de ces actions peut donc être optimisée individuellement, alors que pour la recherche de politique globale, cette séquence d'actions est vue comme un tout et est optimisée d'un bloc. Des méthodes de recherche de politique globale, aussi appelées la recherche directe de politique, sont présentées dans la section 2.4.

Cette section est centrée sur l'apprentissage par renforcement et donc sur les politiques séquentielles. Elle présente d'abord les méthodes de prise de décision séquentielle qui sont les premières méthodes de résolution de l'apprentissage par renforcement (Section 2.3.1). Elle présente ensuite de manière non exhaustive des méthodes de recherche de politiques sous forme d'un réseau de neurones qui sont maintenant utilisées grâce à l'apprentissage profond

pour réaliser de l'apprentissage par renforcement profond (Section 2.3.2). Enfin, elle présente deux variantes de l'apprentissage par renforcement profond qui essaient de pallier plusieurs problèmes : l'apprentissage par renforcement basé sur un modèle (Section 2.3.3) et le méta-apprentissage par renforcement (Section 2.3.4).

2.3.1 Prise de décision séquentielle

L'apprentissage par renforcement prend son origine dans des recherches qui étudient comment les animaux apprennent des comportements (Thorndike, 1898 ; Skinner, 1938). Il se base sur le principe que l'animal est doté d'un système de punition et de récompense qui le conditionne à éviter certains comportements et à en promouvoir d'autres, de la même manière qu'un chien peut apprendre un nouveau tour grâce à des friandises. La retranscription de cette approche dans le cadre de l'informatique et plus particulièrement de l'apprentissage date de la fin des années 70 et du début des années 80 (Kaelbling et al., 1996), avec une formalisation largement répandue aujourd'hui par Sutton et Barto (1998).

L'objectif de l'apprentissage par renforcement est de trouver des comportements uniquement grâce à l'interaction de l'agent avec l'environnement et le retour d'une récompense qui mesure la qualité du comportement pour la tâche souhaitée. La boucle d'interaction est la suivante : un agent dans un état $s \in S$ effectue une action $a \in A$ et récupère un nouvel état $s' \in S$ et une récompense $r \in \mathbb{R}$. Le but est de maximiser la somme des récompenses récoltées tout au long de la séquence d'actions $a_{1:t}$.

Les premiers algorithmes d'apprentissage par renforcement considèrent des problèmes où l'espace des actions A et des états S sont tous deux discrets. Ceci permet d'utiliser des méthodes de programmation dynamique (Bellman, 1966 ; Sutton et Barto, 1998) pour remplir soit une table de valeur $Q(s, a)$ qui apprend l'espérance de récompense (en supposant que l'agent choisira des actions optimales par la suite), soit une table $V(s)$ qui, pour chaque état s, estime l'espérance de récompense si l'agent choisit des actions optimales à partir de celui-ci.

Ce problème s'apparente au problème du bandit manchot (Berry et al., 1985), qui correspond à maximiser la récompense obtenue à partir d'un nombre fini d'actions à la différence qu'il n'y a pas d'information explicite d'état s de l'environnement. Les bandits contextuels (T. Lu et al., 2010) optimisent la prise de décision à l'aide d'un contexte de l'environnement, ce qui peut s'apparenter à un état de l'environnement. La principale différence avec l'apprentissage par renforcement est que les algorithmes de bandit n'ont pas de phase d'apprentissage ; ils ne s'intéressent qu'à optimiser exploration et exploitation pour obtenir le maximum de récompense, le plus souvent pour un horizon court. Autrement dit, un algorithme de bandit est une politique et non pas un algorithme d'apprentissage de politique.

Dans l'hypothèse où la fonction de transition $T(s'|s, a)$ (la fonction qui donne la probabilité de passer d'un état s à un état s' en effectuant l'action a) est connue, les tables $Q(s, a)$ et $V(s)$ peuvent être utilisées pour optimiser l'action a à prendre dans un état s. Si la fonction de transition n'est pas connue, il existe deux approches : les méthodes avec modèle (qui essaient d'apprendre cette fonction de transition) (Section 2.3.3) et les méthodes sans modèle qui apprennent uniquement une politique pour choisir les actions.

Les méthodes sans modèle apprennent un critique qui estime la fonction de valeur $V(s)$ et un acteur qui optimise l'action à prendre pour optimiser la prédiction du critique. La méthode *Adaptive Heuristic Critic* (Barto et al., 1983) est l'une des premières utilisant la famille d'algorithmes $TD(\lambda)$ pour résoudre ce problème. Une évolution de cet algorithme est le *Q-learning* (Watkins et al., 1992), qui, au lieu d'estimer $V(s)$, estime $Q(s, a)$. L'un des principaux avantages est que l'action a est une entrée de la fonction, ce qui permet de

choisir une action de façon plus gloutonne. Cette méthode pose le problème de trouver un équilibre entre exploration et exploitation, mais est globalement considérée comme plus simple à mettre en œuvre. De plus, dans l'hypothèse où lors de l'entraînement chaque paire (s, a) est suffisamment visitée, la convergence de $Q(s, a)$ vers sa valeur optimale ne dépend pas de la méthode d'exploration de l'environnement, contrairement à la fonction de valeur $V(s)$ qui est beaucoup plus sensible à la façon dont les états sont visités.

Une avancée importante pour généraliser l'apprentissage par renforcement à des problèmes continus est l'algorithme REINFORCE (pour "*REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility*") (Williams, 1992), qui utilise la descente de gradient pour mettre à jour les poids du réseau de neurones. REINFORCE est uniquement applicable dans le cas d'une récompense immédiate et non pas d'une optimisation de la récompense à long terme, ce qui en fait un algorithme de recherche directe de politique.

L'algorithme *Episodic Natural Actor Critic* (eNAC) (Peters et al., 2008) apporte comme modifications : (1) l'utilisation de gradient naturel (c'est-à-dire qui prend en compte la géométrie de l'espace des paramètres) pour homogénéiser les possibles différences d'échelles entre les différentes dimensions des paramètres et (2) l'utilisation d'une architecture acteur-critique permettant de résoudre des problèmes d'apprentissage par renforcement avec une récompense à long terme.

Ces algorithmes utilisent comme exploration la perturbation des actions (c'est-à-dire, le plus souvent, un bruit gaussien autour de l'action prédite). Ceci pose non seulement des problèmes de sûreté en robotique (car des actions à haute fréquence peuvent endommager les actionneurs) mais aussi une assez mauvaise exploration (Rückstiess et al., 2010). Pour pallier ce problème, l'algorithme *Policy Learning by Weighting Exploration with the Returns* (PoWER) (Kober et al., 2009) propose de perturber non pas les actions, mais les paramètres de la politique. De plus, PoWER n'utilise plus une estimation du gradient, mais une moyenne pondérée des récompenses, ce qui le ramène à une recherche directe de politique.

L'algorithme *Policy Improvement with Path Integrals* (PI^2) (Theodorou et al., 2010) s'inspire du contrôle par intégrale de chemin généralisé pour reformuler l'étape de mise à jour des paramètres de PoWER. Cette modification lui permet en particulier d'être moins contraint sur la fonction de récompense et le rend en pratique un ordre de grandeur plus rapide en termes d'interactions avec l'environnement.

Il faudra attendre les succès en apprentissage profond pour que des méthodes d'apprentissage par renforcement profond se développent. L'une des premières est l'algorithme de *Deep Q-Network* (DQN) (Mnih, Kavukcuoglu, Silver, Rusu et al., 2015) qui apprend à jouer à des jeux Atari 2600 directement à partir des pixels et non plus de caractéristiques définies à l'avance. Bien qu'impressionnant, DQN n'est applicable qu'à des problèmes où les actions sont finies et discrètes.

2.3.2 Apprentissage par renforcement profond

Pour des environnements plus complexes et où, en particulier, l'espace des actions est continu (par exemple, pour les commandes en position d'un robot), l'apprentissage de comportements est possible via une politique sous forme d'un réseau de neurones profond. Cette section présente les algorithmes d'apprentissage par renforcement profond qui utilisent un gradient pour apprendre cette politique. Il existe aussi des méthodes de recherche directe de politique qui proviennent le plus souvent du calcul évolutionniste et des stratégies d'évolution (Section 2.4).

Les algorithmes d'apprentissage par renforcement profond apprennent une politique (acteur) sous forme $\pi_\theta(a|s)$ où θ représente les poids du réseau de neurones profond. L'idée est d'optimiser la politique de façon à maximiser l'espérance de récompense :

$$J(\theta) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q^\pi(s,a) \pi_\theta(a|s)$$

où $d^\pi(s)$ est la distribution d'états pour la politique π_θ. Pour ce faire, les poids du réseau de neurones sont optimisés itérativement par ascension de gradient de la façon suivante :

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t).$$

La difficulté est d'estimer ce gradient $\nabla J(\theta_t)$. Le théorème de gradient de politique (Sutton, McAllester et al., 1999) montre que la dérivée de la fonction de récompense ne dépend que de la dérivée de la politique et pas de la distribution d'état qui lui est liée :

$$\nabla J(\theta) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s).$$

Une question est de savoir si la politique doit être entraînée avec des données provenant d'interactions de l'acteur avec l'environnement, ce qui est appelé l'apprentissage en ligne (*on-policy* en anglais) (Section 2.3.2.1), ou avec des données provenant d'une autre politique, ce qui est appelé l'apprentissage hors ligne (*off-policy* en anglais) (Section 2.3.2.2).

Pour les méthodes d'apprentissage en ligne, le gradient peut se réécrire :

$$\nabla J(\theta) = \mathbb{E}_\pi[Q^\pi(s,a) \nabla_\theta \log(\pi_\theta(a|s))].$$

Pour les méthodes d'apprentissage hors ligne, le gradient peut se réécrire :

$$\nabla J(\theta) = \mathbb{E}_\beta \left[\frac{\pi_\theta(a|s)}{\beta(a|s)} Q^\pi(s,a) \nabla_\theta \log(\pi_\theta(a|s)) \right].$$

Comme la politique correspond à un réseau de neurones paramétré par ses poids, le gradient peut être calculé grâce à des méthodes de différentiation automatique (Baydin et al., 2018).

2.3.2.1 Apprentissage en ligne (*on-policy*)

Les algorithmes d'apprentissage en ligne ont tendance à être plus faciles à implémenter, plus robustes aux hyperparamètres, mais souffrent d'une exploration moindre et nécessitent plus de données, car celles-ci sont utilisées qu'une seule fois avant d'être jetées.

Quelques algorithmes d'apprentissage en ligne parmi les plus connus sont :

— *Trust Region Policy Gradient* (TRPO) (Schulman, Levine et al., 2015), dont l'idée principale est de borner la mise à jour de la politique de façon à éviter l'explosion de gradient (pour respecter cette contrainte, TRPO utilise une distance de Kullback-Leibler nécessitant une approximation du second ordre, ce qui le rend peu efficace pour des problèmes de hautes dimensions) ;

— *Actor Critic using Kronecker-Factored Trust Region* (ACKTR) (Y. Wu et al., 2017), qui adapte l'optimisation du second ordre de TRPO à l'aide de l'approximation de courbure *Kronecker-factored*, rendant cette technique de région de confiance plus simple à utiliser ;

— *Proximal Policy Optimization* (PPO) (Schulman, Wolski et al., 2017), qui est développé plus en détail dans le paragraphe suivant.

Proximal Policy Optimization. PPO vise à obtenir la simplicité d'utilisation d'une méthode de *Q-learning* comme DQN (Mnih, Kavukcuoglu, Silver, Rusu et al., 2015) et la robustesse de TRPO (Schulman, Levine et al., 2015). En effet, *Q-learning* souffre de problèmes de robustesse, d'efficacité d'utilisation des données et n'est pas toujours capable de résoudre des problèmes simples. TRPO est compliqué à mettre en œuvre et n'est pas applicable avec toutes les architectures d'apprentissage profond, comme le *dropout* ou le partage de paramètres entre l'acteur et le critique.

PPO approxime l'optimisation du second ordre de TRPO par une optimisation du premier ordre qui limite la mise à jour de la politique lorsque celle-ci devient supérieure à un hyperparamètre. Ceci permet à PPO de profiter de la stabilité de mise à jour de l'acteur tout en pouvant utiliser une architecture de réseaux de neurones où l'acteur et le critique partagent une partie des paramètres. De plus, PPO ajoute à sa fonction de coût (fonction utilisée pour obtenir le gradient) un terme supplémentaire d'entropie pour améliorer l'exploration (comme suggéré par Mnih, Badia et al. (2016a)).

PPO est aujourd'hui largement utilisé principalement du fait de sa simplicité et de ses hyperparamètres robustes. Il a fait l'objet de nombreuses applications, par exemple à de la gestion de chaîne de ravitaillement (M. Zhang et al., 2022), l'aérospatial (W. Chen et al., 2023), la finance (Lin et al., 2021), l'énergie (Pinciroli et al., 2021), les jeux vidéo (Kristensen et al., 2020), la conduite autonome (Guan et al., 2020), et la robotique (Sun et al., 2023). C'est pourquoi dans cette thèse nous avons décidé de l'utiliser comme méthode de référence de l'apprentissage par renforcement profond.

2.3.2.2 Apprentissage hors ligne (*off-policy*)

Les algorithmes d'apprentissage hors ligne stockent le plus souvent les données dans un tampon de mémoire (*replay buffer*) qui peuvent donc être réutilisées à plusieurs reprises. Cette réutilisation des données leur permet d'une part, d'être plus efficaces en termes d'échantillonnage et, d'autre part, de pouvoir utiliser n'importe quelle méthode pour collecter les données. Par contre, elles sont généralement considérées comme plus complexes à paramétrer.

Quelques algorithmes d'apprentissage hors ligne parmi les plus connus sont :

— *Deep Deterministic Policy Gradient* (DDPG) (Lillicrap et al., 2015b), qui étend les idées de DQN aux actions continues ;
— *Advantage Actor-critic* (A2C) et *Asynchronous Advantage Actor-critic* (A3C) (Mnih, Badia et al., 2016b), qui diffèrent du fait qu'ils apprennent la fonction de valeur $V(s)$ au lieu de la Q-fonction $Q(s, a)$;
— *Twin Delayed Deep Deterministic Policy Gradient* (TD3) (Fujimoto et al., 2018), qui règle certains problèmes d'instabilités de DDPG en (1) entraînant deux critiques Q-fonctions pour prendre le minimum des deux, en (2) mettant à jour l'acteur moins souvent que les Q-fonctions et (3) en ajoutant du bruit dans les actions pour diminuer la surexploitation des Q-fonctions ;
— *Soft Actor-Critic* (SAC) (Haarnoja et al., 2018), qui prend le meilleur de chaque en utilisant un tampon de mémoire comme DDPG, une méthode Acteur-Critique comme A2C, et utilise une régularisation entropique pour améliorer l'exploration comme PPO avec une adaptation automatique du paramètre de température pour équilibrer exploration et exploitation en fonction de l'environnement.

2.3.3 Apprentissage par renforcement profond basé sur un modèle

L'ensemble de ces méthodes est considéré comme sans modèle de la dynamique de l'environnement. C'est-à-dire qu'elles ne modélisent qu'une sous-partie de l'environnement avec la fonction de récompense, la fonction de valeur $V(s)$ ou la Q-fonction $Q(s, a)$, mais pas la fonction de transition $T(s'|s, a)$. Une autre branche de l'apprentissage par renforcement profond, dite basée sur un modèle, s'intéresse justement à apprendre et exploiter une telle fonction (Moerland et al., 2023).

L'idée générale est qu'en apprenant la fonction de transition, l'agent peut s'entraîner de façon "gratuite" avec une copie de l'environnement sans nécessiter d'interactions supplémentaires avec l'environnement réel. Dans la pratique, le modèle appris n'est jamais parfait et l'agent risque de sur-apprendre et donc de mal généraliser à l'environnement réel. Une difficulté supplémentaire est de récupérer des données qui couvrent les transitions intéressantes pour la réalisation de la tâche.

L'algorithme décrit par D. Ha et al. (2018) propose d'entraîner un auto-encodeur variationnel (VAE) avec une politique aléatoire pour compresser l'information visuelle de l'environnement dans un espace latent. Il entraîne ensuite un réseau de neurones récurrent pour prédire la fonction de transition dans l'espace latent, puis optimise une politique sous la forme d'un réseau de neurones linéaire en utilisant CMA-ES (Hansen et al., 2003) (un algorithme d'optimisation boîte-noire présenté dans la section 2.4.1.2) et le modèle de l'environnement appris. L'entraînement peut aussi se faire sous forme itérative, c'est-à-dire que l'algorithme peut boucler en réutilisant la politique apprise pour récupérer de nouvelles données, apprendre un meilleur modèle et puis une meilleure politique, jusqu'à ce qu'une politique satisfaisante soit apprise. Il montre son succès à apprendre des politiques à partir d'images sur deux environnements de jeux vidéos (*CarRacing-v0* et *VizDoom : Take Cover*) en étant plus performant que DQN (Mnih, Kavukcuoglu, Silver, Rusu et al., 2015) et A3C (Mnih, Badia et al., 2016b).

D'autres méthodes existent, par exemple :

— *Probabilistic Inference for Learning COntrol* (PILCO) (Deisenroth et Rasmussen, 2011), qui modélise l'environnement avec des processus gaussiens pour avoir une estimation de l'incertitude et qui utilise une optimiseur basé sur le gradient pour obtenir une politique, mais qui ne passe pas aussi bien à l'échelle que des réseaux de neurones profonds en termes de dimensionalité du vecteur de description de l'environnement ;

— *Black-box Data-efficient Robot Policy Search* (Black-DROPS) (Chatzilygeroudis, Rama et al., 2017), qui modélise l'environnement comme PILCO mais qui utilise CMA-ES (Hansen et al., 2003) pour optimiser une politique, ce qui permet entre autres de pouvoir utiliser n'importe quelle fonction de récompense ;

— les travaux de Nagabandi, Kahn et al. (2018), qui combinent l'apprentissage d'un modèle avec un réseau de neurones et de la commande prédictive ;

— *Probabilistic Ensembles with Trajectory Sampling* (PETS) (Chua et al., 2018), qui apprend un ensemble de modèles avec des réseaux de neurones pour obtenir une incertitude permettant d'utiliser une méthode de commande prédictive avec propagation de l'incertitude.

Les méthodes d'apprentissage par renforcement profond basé sur un modèle combinent le plus souvent un apprentissage non supervisé ou semi-supervisé pour apprendre un modèle et une méthode d'optimisation pour obtenir un contrôle. Elles s'éloignent donc de l'apprentissage par renforcement (c'est-à-dire l'apprentissage se basant uniquement sur le retour d'un signal de récompense) et ne s'y apparentent que pour la formulation séquentielle en paire d'action et d'état, la communauté et les problèmes de référence.

Une réflexion intéressante à avoir pour la robotique en particulier est que les simulateurs physiques sont déjà un modèle de la fonction de transition réelle. Ainsi, l'apprentissage par renforcement profond sans modèle qui utilise un simulateur pour apprendre un comportement sur un robot réel peut s'apparenter à une méthode d'apprentissage basée sur un modèle. La différence est que le modèle est donné et non appris.

2.3.4 Méta-apprentissage

L'un des problèmes avec l'apprentissage par renforcement profond est que l'apprentissage redémarre de zéro pour chaque nouveau problème ou tâche à résoudre. C'est une utilisation des données peu efficace. Par exemple, dans le cas d'un robot qui apprend à pousser, tirer ou saisir des objets, les différentes tâches nécessitent un contrôle du bras du robot dont l'apprentissage peut être factorisé avant d'être spécialisé. Dans ce sens, l'apprentissage par renforcement basé sur un modèle peut aussi réutiliser le modèle pour apprendre plusieurs comportements. Une limitation est que bien souvent le modèle est spécialisé dans les interactions qu'il a rencontrées et généralise mal.

Pour permettre d'apprendre plusieurs problèmes sans redémarrer de zéro, le méta-apprentissage (c'est-à-dire l'apprentissage de l'apprentissage) (Hospedales et al., 2021) permet de factoriser l'apprentissage pour accélérer l'entraînement. Nous pouvons différencier deux approches selon que le méta-apprentissage consiste en un modèle ou un ensemble de poids.

Méta-modèle. Des méthodes apprennent à un méta-réseau de neurones à prendre en entrée une description de tâche pour renvoyer un autre réseau de neurones (par exemple, l'architecture ou les poids des connexions) spécialisé pour la tâche. Un exemple de tel algorithme est RL^2 (Duan et al., 2016) qui entraîne un réseau de neurones récurrent (lent) pour initialiser le poids d'un second réseau de neurones (rapide). La même méthode est développée indépendamment dans les travaux de J. X. Wang et al. (2016).

Méta-poids. Des méthodes optimisent le poids du réseau de neurones de façon à ce qu'ils s'adaptent rapidement à un changement de l'environnement. *Model-Agnostic Meta-Learning* (MAML) (Finn et al., 2017) optimise les poids du réseau de neurones de façon à ce qu'un (ou quelques) pas de descente de gradient permettent de les optimiser pour les adapter à un nouveau problème. Ceci lui permet d'obtenir de meilleures performances que l'état de l'art dans une tâche de classification en quelques essais (*few shot classification*) et d'accélérer l'entraînement de méthodes d'apprentissage par renforcement profond sans modèle. Cette idée est aussi adaptée à une méthode basée sur un modèle pour l'adaptation du contrôle d'un robot quadrupède (Nagabandi, Clavera et al., 2018 ; Kaushik et al., 2020 ; Anne, Wilkinson et al., 2021) [8].

Stadie et al. (2018) proposent $E-RL^2$ et E-MAML, deux extensions respectives des travaux présentés ci-dessus qui entraînent explicitement l'algorithme de méta-apprentissage pour qu'il optimise la distribution d'échantillonnage utilisée par le modèle rapide afin d'en améliorer l'exploration. Ces deux extensions apprennent plus rapidement et couvrent plus efficacement l'espace des états des environnements considérés.

Les méthodes de méta-apprentissage améliorent l'efficacité en échantillonnage mais restent des méthodes complexes à mettre en œuvre et qui nécessitent une importante puissance de calcul.

8. Comme présenté dans la section 1.3, nous avons décidé de ne pas plus élaborer sur ces deux travaux utilisant du méta-apprentissage pour l'adaptation de la locomotion d'un robot quadrupède.

2.3.5 Conclusion sur l'apprentissage par renforcement profond

Les méthodes d'apprentissage par renforcement profond sont aujourd'hui des méthodes matures avec d'importantes applications en robotique. Plusieurs travaux ont ainsi montré que l'apprentissage par renforcement profond pouvait remplacer les contrôleurs classiques : PPO pour le quadrupède Minitaur (Tan, T. Zhang et al., 2018) et TRPO pour le quadrupède ANYmal (Hwangbo et al., 2019). TRPO et PPO sont encore utilisés pour apprendre des comportements de marche robustes pour des environnements réels et accidentés avec ANYmal (J. Lee et al., 2020 ; Miki et al., 2022). En plus de l'apprentissage par renforcement profond, ces travaux utilisent une phase de robustification des politiques via un apprentissage supervisé par imitation des politiques privilégiées apprises en simulation.

Dans le cas du contrôle de drones autonomes, Song et al. (2023) montrent aussi qu'une méthode d'apprentissage par renforcement profond (une version modifiée de PPO) obtient de meilleures performances que deux méthodes de contrôle classique pour de la course de drone en salle (*Trajectory Tracking* (Foehn et al., 2021) et *Contouring Control* (Romero et al., 2022)). La même méthode permet d'obtenir des performances similaires aux meilleurs experts humains et a même démontré le record du plus court temps de vol pour la course (Kaufmann et al., 2023).

Ce n'est donc pas un manque de performance qui nous pousse à chercher une alternative, mais principalement leur pénibilité d'utilisation.

2.4 Stratégies d'évolution pour l'apprentissage par renforcement

Les méthodes de recherche directe de politique optimisent la politique sans nécessiter de gradient. Elles sont le plus souvent appliquées à des problèmes dits boîte noire, c'est-à-dire que l'algorithme n'a accès qu'à la valeur de la fonction de récompense (ou *fitness*).

2.4.1 L'état de l'art

2.4.1.1 La méthode de l'entropie croisée

La méthode de l'entropie croisée (CEM) (Rubinstein et al., 2004a ; De Boer et al., 2005) est une méthode d'échantillonnage préférentiel de type Monte-Carlo de recherche directe de politique. Elle a été utilisée en robotique, par exemple pour de la planification de mouvement (Kobilarov, 2012). Chaque itération se compose de deux étapes : une étape d'échantillonnage de solutions selon la distribution courante et une étape de mise à jour de la distribution courante à l'aide des échantillons, en se basant sur la minimisation de l'entropie croisée et de la divergence de Kullback-Leibler pour se rapprocher d'une distribution autour de la solution optimale. Cette méthode se range dans la catégorie des méthodes statistiques des mathématiques, mais a de fortes ressemblances avec les stratégies d'évolution du calcul évolutionniste (Stulp et al., 2013).

2.4.1.2 *Covariance Matrix Adaptation Evolution Strategy*

Le calcul évolutionniste (Back et al., 1997) est un champ de recherche qui s'intéresse particulièrement à l'optimisation. Les algorithmes d'évolution se divisent en nombreuses catégories, mais partagent le plus souvent le même principe de faire évoluer des solutions en effectuant

des mutations ou croisements pour se diriger vers un optimum. Ils peuvent ainsi résoudre le problème de recherche directe de politique pour des environnements boîte noire, comme c'est le cas avec un robot réel.

Les stratégies d'évolution (Rechenberg, 1972 ; Schwefel, 1981 ; Beyer et al., 2002) forment une sous-catégorie d'algorithmes d'évolution qui s'intéressent particulièrement à l'adaptation de la mutation. Plus formellement, une stratégie d'évolution adapte une distribution gaussienne en échantillonnant une population à partir de celle-ci puis en l'utilisant pour sonder l'espace de recherche afin de mettre à jour les paramètres de la distribution.

La méthode de *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) (Hansen et al., 2003) est aujourd'hui considérée comme l'état de l'art pour l'optimisation d'une fonction boîte noire. Elle se base, tout comme CEM, sur la mise à jour d'une distribution de solutions dans le but d'échantillonner des individus qui se rapprochent le plus possible de la solution optimale. Elle a deux différences clés. D'une part, CMA-ES ne se base que sur le rang des individus, ce qui lui permet d'avoir moins d'hypothèses sur le problème (l'accès à la valeur de la fonction optimisée n'est même pas requis) et d'être plus robuste (si du bruit ne change pas l'ordre des solutions, la méthode aura la même exécution). D'autre part, CMA-ES optimise sa distribution pour maximiser les chances d'amélioration des individus d'une génération à l'autre. C'est-à-dire qu'elle maximise le pas que fait la distribution dans la direction de la distribution optimale. En comparaison, CEM optimise directement sa distribution pour maximiser les chances qu'un individu soit optimal.

CMA-ES est utilisée dans de nombreuses applications [9]. Étant fondamentalement une technique d'échantillonnage, elle se prête difficilement à des problèmes où les solutions possèdent plus de quelques milliers de paramètres. Elle n'est donc pas facilement applicable pour optimiser efficacement les poids d'un réseau de neurones pouvant atteindre plusieurs millions de paramètres. Elle a tout de même été montrée capable d'optimiser, par exemple, les poids d'un "petit" réseau de neurones (moins de 10 000 paramètres) pour adapter les gains d'une tâche de poursuite de chemin avec un robot (Hill et al., 2020) et pour le contrôle d'un agent dans deux jeux vidéos (D. Ha et al., 2018) (moins de 1 000 paramètres).

2.4.2 Similarité avec l'apprentissage par renforcement (non profond)

Stulp et al. (2013) ont réalisé un travail de comparaison entre les méthodes d'apprentissage par renforcement non profond (REINFORCE, eNAC, PoWER et PI^2) et une stratégie d'évolution $(\mu_W, \lambda) - ES$ (c'est-à-dire une stratégie d'évolution où, à chaque itération une combinaison pondérée des μ meilleures solutions parmi les λ solutions candidates est utilisée pour mettre à jour la distribution) sur laquelle se basent CMA-ES et CEM. La comparaison est possible car la somme des récompenses d'un épisode d'apprentissage par renforcement peut se voir comme la *fitness* des stratégies d'évolution. Deux autres parallèles sont tirés : d'une part, la perturbation des paramètres qu'effectue PoWER et PI^2 correspond aux mutations qu'effectuent les stratégies d'évolution et d'autre part, la mise à jour des poids du réseau de neurones correspond à la phase de mise à jour des paramètres de la distribution gaussienne. En appliquant deux modifications à PI^2 (rendre le bruit d'exploration constant au cours d'un épisode et remplacer le moyennage temporel par la somme des récompenses de l'épisode), les auteurs montrent que PI^2 et $(\mu_W, \lambda) - ES$ sont deux versions du même algorithme.

9. Voici une liste non exhaustive de travaux utilisant CMA-ES maintenue par les auteurs jusqu'en 2009 : http://www.cmap.polytechnique.fr/~nikolaus.hansen/cmaapplications.pdf

Les différents algorithmes sont ensuite comparés et les résultats montrent que le plus simple algorithme de stratégies d'évolution $(\mu_W, \lambda) - ES$ converge plus vite et vers une meilleure solution que les algorithmes plus complexes d'apprentissage par renforcement. Les auteurs l'expliquent par le fait que dans leur comparaison, la variable optimisée par les stratégies d'évolution n'est pas directement une séquence d'actions mais une politique paramétrée à l'aide de *Dynamic Movement Primitives* (DMPs) (Ijspeert et al., 2013). Les DMPs simplifient le problème en réduisant la dimensionnalité du problème et en optimisant chaque dimension séparément.

Une conclusion de ces travaux est que le choix d'une bonne représentation du problème permet de faciliter celui-ci. Le prix à payer est la restriction de l'espace des comportements possibles. L'apprentissage profond vient changer la donne grâce au développement fin des années 2000 de matériel permettant d'optimiser les paramètres d'un réseau de neurones profond et donc de fournir de bonnes approximations pour des fonctions de grandes dimensions (G. Hinton et al., 2012 ; Amodei et al., 2018).

2.4.3 Comparaison avec l'apprentissage par renforcement profond

Salimans et al. (2017) comparent des stratégies d'évolution avec les méthodes d'apprentissage par renforcement profond. Ils utilisent OpenAI-ES, une méthode de la famille des stratégies d'évolution naturelle (Wierstra et al., 2014). Celles-ci utilisent le gradient naturel (c'est-à-dire un gradient qui tient compte de la géométrie de l'espace des paramètres), ce qui permet, entre autres, d'être plus résistant aux minima locaux et aux oscillations. OpenAI-ES obtient les mêmes performances que TRPO (Schulman, Levine et al., 2015) sur les environnements de robotique continue MuJoCo (Todorov et al., 2012) de référence provenant de *OpenAI Gym* (Brockman et al., 2016) en moins de 10 minutes de temps réel, et est aussi compétitive sur les environnements de référence Atari provenant de *OpenAI Gym* en moins d'une heure d'entraînement.

OpenAI-ES requiert, par contre, le plus souvent, plus d'interactions avec l'environnement (par exemple, presque 10 fois plus que TRPO sur les environnements les plus durs). Ceci s'explique par le fait qu'elle a accès à moins d'informations. OpenAI-ES a, par contre, d'autres avantages comme le fait d'être beaucoup plus simple et facilement parallélisable. C'est cette fonctionnalité qui lui permet d'être plus rapide lorsque nous considérons le temps de calcul réel. De plus, comme elle optimise directement la politique, elle est invariante à la fréquence de discrétisation de l'environnement (c'est-à-dire le pas de temps entre chaque action). Au contraire, il a été montré que des méthodes d'apprentissage par renforcement comme DQN en sont sensibles dans les environnements Atari (Braylan et al., 2015). De plus, comme elle ne nécessite pas de gradient, il n'y a pas de problème d'attribution de crédit ou de disparition du gradient. Elle peut donc s'appliquer à des durées de comportements longues.

Majid et al. (2023) offre une revue détaillée de la comparaison entre les méthodes de stratégies d'évolution et d'apprentissage par renforcement profond. Cette revue met en lumière d'autres méthodes de stratégies d'évolution :

— *Novelty Search ES* (NS-ES) (Conti et al., 2018) y ajoute des techniques de l'algorithme *Novelty Search* (Lehman et al., 2011a) (présenté dans la section 2.5.1.1) et aussi de Qualité-Diversité (présentée dans la section 2.5.1.2) pour améliorer l'exploration et éviter les minima locaux ;

— *Trust Region Evolution Strategies* (TRES) (G. Liu et al., 2019) optimise une fonction auxiliaire utilisant une région de confiance pour garantir une optimisation minimale et permettant de réutiliser des données, et montre qu'elle est plus efficace en données que les stratégies d'évolution de l'état de l'art et obtient une efficacité de même ordre que PPO et TRPO ;
— *Evolution Strategy with Progressive Episode Lengths* (PEL) (Fuks et al., 2019) définit un curriculum d'apprentissage pour une stratégie d'évolution en accroissant progressivement la longueur des épisodes d'entraînement, et montre que ceci permet d'obtenir de meilleures performances que d'autres stratégies d'évolution pour des jeux Atari.

Il existe aussi des méthodes qui hybrident les stratégies d'évolution et les méthodes qui utilisent le gradient :

— CEM-RL (Pourchot et al., 2019) combine CEM (Rubinstein et al., 2004b) avec TD3 (Fujimoto et al., 2018) en faisant évoluer une population d'acteurs grâce à CEM et en utilisant TD3 pour entraîner le critique, obtenant ainsi de meilleurs scores que TD3 et CEM sur des environnements de référence OpenAI ;
— *Evolved Policy Gradients* (Houthooft et al., 2018) fait évoluer grâce à CMA-ES (Hansen et al., 2003) la fonction de coût utilisée par une méthode de descente de gradient, permettant d'entraîner le modèle plus rapidement ;
— *Multi-Objective CMA-ES* (D. Chen et al., 2020) combine une modification multi-politique de SAC (Haarnoja et al., 2018) avec CMA-ES (Hansen et al., 2003) pour résoudre des problèmes multi-objectifs et trouver des fronts de Pareto ;
— *Fine-tune DRL* (Bruin et al., 2020) combine CMA-ES (Hansen et al., 2003) avec DQN (Mnih, Kavukcuoglu, Silver, Rusu et al., 2015) (pour les problèmes discrets) et DDPG (pour les problèmes continus) en utilisant CMA-ES pour optimiser les dernières couches du réseau de neurones acteur, et obtient de meilleurs scores que les méthodes DQN et DDPG seules tout en étant plus efficace en échantillonnage.

2.5 La robotique évolutionniste

La robotique évolutionniste (Nolfi et al., 2000 ; Doncieux, Mouret et al., 2011 ; Bongard, 2013 ; Doncieux, Bredeche et al., 2015a) est née de la volonté de combiner le concept d'intelligence incarnée (Pfeifer et Bongard, 2006) avec le calcul évolutionniste (Eiben et al., 2003). Le calcul évolutionniste s'inspire de principes biologiques pour développer des méthodes qui résolvent des problèmes, souvent d'optimisation. Les stratégies d'évolution présentées précédemment en sont un exemple. Un constat en robotique est que souvent chaque aspect du robot est étudié et conçu séparément (Siciliano et al., 2008). Par exemple, pour réaliser une tâche d'exploration, un ingénieur peut : prendre un robot déjà conçu ; construire une carte de l'environnement avec un algorithme de *Self Localization and Mapping* (SLAM) (Thrun, 2002) ; optimiser le déplacement du robot avec un algorithme de planification de trajectoire (Siegwart et al., 2011) ; et déplacer le robot grâce à un contrôleur de locomotion (Schulman, Wolski et al., 2017). La modularisation de ces différents composants permet une conception plus aisée en partie grâce à une vérification indépendante, mais peut rendre l'agrégation des différents composants sous-optimale.

Le concept d'intelligence incarnée s'intéresse justement à étudier des robots, l'environnement et l'interaction entre tous les composants comme un tout. Les êtres vivants sont un bon exemple d'une intelligence incarnée, ce qui amène à s'intéresser aux processus biologiques qui en ont permis l'évolution.

La robotique évolutionniste s'inspire donc des algorithmes du calcul évolutionniste pour faire évoluer des robots en suivant le principe de sélection et variation d'individus évalués dans un environnement. Chaque individu est représenté par son génotype (la variable d'optimisation du processus global) qui est décodé pour former l'individu décrit par son phénotype (les caractéristiques de l'individu pouvant donc contenir sa physiologie et sa cognition). Le processus démarre avec une première génération d'individus générée aléatoirement. Chaque individu est évalué en interagissant avec l'environnement ce qui permet de collecter un comportement, une séquence d'état ou juste une *fitness*. Une nouvelle génération d'individus est ensuite créée en sélectionnant les individus à partir de la génération courante grâce à un ensemble de critères (par exemple les individus avec la plus grande *fitness*) et en appliquant un opérateur de variation (par exemple une mutation ou un croisement). Le processus évolue ainsi une série de générations d'individus jusqu'à ce qu'un critère terminal soit atteint (par exemple la résolution du problème ou un budget d'évaluation).

Doncieux, Bredeche et al. (2015b) proposent un état des lieux de la robotique évolutionniste et une discussion sur son avenir. Ils commencent d'abord par présenter une série de points clés.

Les réseaux de neurones comme politique. Pour les mêmes raisons que l'apprentissage profond, les réseaux de neurones sont un bon choix de contrôleur à évoluer, grâce à leur universalité théorique et les nombreux déploiements pratiques.

La diversité comportementale pour guider l'exploration. L'utilisation de la *fitness* pour diriger l'exploration des solutions n'est pas toujours le meilleur choix, surtout dans les environnements avec une *fitness* éparse ou trompeuse (un résultat présenté dans la section 2.5.1.1).

La pression de sélection. La modélisation de la pression de sélection est au moins aussi importante que les représentations choisies pour les individus (c'est-à-dire le génotype, le phénotype et le lien entre les deux) (Doncieux et Mouret, 2014). En effet, une pression de sélection trop forte, par exemple uniquement basée sur la *fitness*, peut entraîner une convergence précoce des solutions vers un minimum local. À l'opposé, une pression de sélection moins forte, par exemple garder une diversité de solutions, permet d'obtenir à la fin une meilleure solution globale.

Le passage à la réalité. Le plus souvent, les robots, leur contrôleur ou les deux sont évolués en simulation, ce qui pose le problème de passer de la simulation à la réalité. En effet, un simulateur ne sera jamais parfait. Une robustification par ajout de bruit est souvent trop conservatrice, l'apprentissage d'une fonction de transfert ne peut aller au-delà de ce que le simulateur permet et apprendre directement sur des robots réels est bien souvent trop coûteux.

Ils proposent ensuite une liste de problèmes encore irrésolus.

Application à des problèmes concrets. Les quelques applications concrètes de la robotique évolutionniste à des problèmes réels n'utilisent les méthodes évolutionnistes que pour une partie de la méthode et non pas encore comme un tout. Par exemple, c'est le cas pour générer des comportements d'un essaim de drones (Hauert et al., 2009) ou pour remporter la RoboCup 2014 (MacAlpine et al., 2015). Les idées de la robotique évolutionniste ont aussi des potentiels concrets en animation, avec par exemple l'apprentissage de mouvements sur des créatures bipèdes où la morphologie musculaire et leur contrôle sont optimisés simultanément à l'aide de CMA-ES (Geijtenbeek et al., 2013).

Une évolution organique. Comprendre l'ensemble des mécanismes de l'évolution biologique qui rendent le processus viable et savoir comment les retranscrire pour la robotique évolutionniste est toujours une question ouverte (Wagner et al., 1996 ; Gerhart et al., 2007). En particulier, la conception d'opérateurs de variation qui permettent de maximiser les chances qu'une variation d'un individu reste viable. Par exemple, pour la programmation génétique, une variation purement aléatoire du programme a de fortes chances de rompre la syntaxe et donc de "tuer" le nouvel individu. C'est pourquoi, par exemple, les algorithmes de programmation génétique utilisent le plus souvent une représentation avec des arbres syntaxiques. Ce qui leur permet d'obtenir plus facilement des variations qui préservent la syntaxe du langage. Les avancées récentes sur les grands modèles de langage (LLMs) pourraient permettre de guider ces variations en se servant de l'information compressée provenant des grandes bases de données de code présentes sur l'internet (Mouret, 2024).

La place de l'apprentissage. Il est encore difficile de savoir comment intégrer l'apprentissage individuel dans le processus d'évolution de la même manière que les êtres vivants apprennent au cours de leur vie en même temps que les espèces évoluent au cours de millions d'années (G. E. Hinton et al., 1987). Le problème provient surtout de la réduction d'échelle des deux processus dans le cas de la robotique évolutionniste, ce qui les met en compétition.

L'évolution en ligne. La plupart des méthodes font d'abord évoluer un ou des robots dans l'environnement pour apprendre à résoudre un problème et ensuite déploient ces robots en stoppant l'évolution. Cela suppose que l'environnement est statique, ce qui est rarement le cas dans le monde réel. La conception de méthodes d'apprentissage de comportements dans un environnement fluctuant pose deux problèmes : d'une part, il n'y a pas d'état initial constant dans lequel réinitialiser l'agent et d'autre part, des contraintes de sécurité ne peuvent être garanties car l'environnement est inconnu.

Une pression de sélection dirigée par l'environnement. Dans la nature, la pression de sélection est déterminée par la capacité d'un individu à générer une descendance. Elle est donc le fruit d'une dynamique complexe avec l'ensemble des constituants de l'environnement et non pas seulement le résultat d'une performance de *fitness*. L'étude de ce type d'évolutions dirigées par l'environnement est encore peu étudiée mais pourrait permettre de s'émanciper encore plus des désavantages des méthodes basées sur un objectif de performance.

La quête de l'évolution sans fin (open endness). L'évolution naturelle est un processus sans fin. Savoir comment réaliser cela avec des agents artificiels est encore une question ouverte. Y répondre pourrait permettre non seulement une meilleure modélisation et compréhension du processus biologique mais aussi de fournir aux agents artificiels la capacité de toujours pouvoir s'améliorer et s'adapter.

2.5.1 La diversité comme moteur d'exploration

Une majeure partie des méthodes présentées jusqu'ici s'intéresse à maximiser un objectif, comme une récompense ou une *fitness*. Un axe tout à fait différent consiste à abandonner l'objectif pour se concentrer sur la diversité comportementale (Stanley et Lehman, 2015).

2.5.1.1 Diversité comportementale

L'algorithme *Novelty-Search* (Lehman et al., 2008 ; Lehman et al., 2011a) s'éloigne intentionnellement du paradigme d'optimisation d'une fonction d'objectif, de *fitness*, de récompense ou de coût. Il se base sur le constat que ce paradigme peut mener à des comportements

sous-optimaux lorsque le problème est trompeur ou doté de nombreux minima locaux. *Novelty-Search* propose de se détacher d'une optimisation d'une telle fonction pour se tourner vers l'optimisation de la diversité comportementale. L'idée étant qu'un algorithme qui parvient à découvrir un large ensemble de comportements a de fortes chances d'avoir aussi découvert des comportements qui résolvent le problème.

Novelty-Search utilise l'algorithme de neuro-évolution *NeuroEvolution of Augmenting Topologies* (NEAT) (Stanley et Miikkulainen, 2002) en remplaçant la sélection des individus à l'aide d'une fonction de *fitness* par une mesure de nouveauté des comportements. La mesure de nouveauté pour un nouvel individu est sa distance moyenne dans l'espace des comportements à ses k plus proches voisins d'une archive de comportements déjà rencontrés. Cette mesure suppose donc également l'existence d'une distance entre des comportements. L'algorithme construit donc deux choses : (1) une population d'individus sélectionnés pour leur nouveauté et (2) une archive d'individus divers mise à jour lorsqu'un nouvel individu est suffisamment éloigné de ceux déjà présents.

Novelty-Search a été évalué dans sa publication originelle dans un problème d'exploration de labyrinthe, où justement essayer d'optimiser la distance à la position cible est une approche trompeuse avec de nombreux minima locaux. Chaque individu correspond à un robot doté de capteurs sensoriels et d'un réseau de neurones de contrôle, et le comportement correspond à la distance finale atteinte. Pour un labyrinthe simple, *Novelty-Search* est trois fois plus rapide que NEAT dirigé par la *fitness*. Et pour un labyrinthe difficile, *Novelty-Search* réussit à le résoudre 39 fois sur 40. Au contraire, NEAT dirigé par la *fitness* et NEAT avec une sélection aléatoire n'y parviennent que 3 et 4 fois sur 40, respectivement. Optimiser pour couvrir l'espace des comportements (ici l'ensemble du labyrinthe) et non pour atteindre la cible permet de trouver un contrôleur qui atteint la cible. Une seconde expérience pour faire évoluer des contrôleurs de marche (des réseaux de neurones continus et récursifs) pour un robot bipède en simulation montre encore une fois que *Novelty-Search* permet de trouver des contrôleurs traversant en moyenne 4,04 m contre 2,88 m pour NEAT dirigé par la *fitness*.

Novelty-Search a été appliqué à de nombreux autres problèmes, par exemple les réseaux de neurones avec plasticité (Risi et al., 2010), un essaim de robots (Gomes et al., 2013), l'apprentissage non supervisé (Naredo et al., 2013) et la génération de niveaux de jeu vidéo (Liapis et al., 2015).

Doncieux, Laflaquière et al. (2019) affirment que *Novelty-Search* correspond à une recherche aléatoire uniforme dans l'espace des comportements, ce qui la rend intéressante étant donné que cet espace n'est pas directement accessible. La difficulté principale de *Novelty-Search* est la conception de cet espace de comportements, qui peut avoir un impact important sur la convergence de la population. Choisir une fonction de *fitness* directement liée à la tâche à résoudre peut sembler plus facile.

2.5.1.2 Qualité-Diversité

La diversité comportementale seule ne permet pas de résoudre tous les problèmes. Par exemple, un agent qui apprendrait à générer des images pourrait essayer de couvrir l'espace autant que possible en maximisant la diversité sans jamais générer autre chose que du bruit. L'association de la recherche de performance dirigée vers un objectif avec de la diversité a été baptisée l'illumination et plus couramment la Qualité-Diversité (QD) (Mouret et Doncieux, 2009 ; Mouret et Doncieux, 2012 ; Doncieux et Mouret, 2014 ; Pugh, Soros, Szerlip et al., 2015 ; Mouret et Clune, 2015 ; Pugh, Soros et Stanley, 2016). Elle consiste à trouver un large et divers ensemble de solutions performantes pour un problème.

Novelty Search with Local Competition (NSLC) (Lehman et al., 2011b) est le premier algorithme de qualité-diversité. Il s'inspire de *Novelty-Search* en ajoutant un objectif local de performance. NSLC s'inspire du principe de niche écologique en biologie. Par exemple, le guépard, le serpent et la fourmi ont des modes de locomotion bien différents. Ils n'ont pas tous convergé vers le mode de locomotion le plus rapide du guépard, car ils ne sont pas en compétition dans les mêmes niches écologiques. L'idée de NSLC est donc de modéliser une compétition locale dans l'espace des comportements. Pour ce faire, NSLC modifie *Novelty-Search* lors de l'étape de calcul de la nouveauté d'un individu (via une comparaison à ses k plus proches voisins) en ajoutant un second objectif de *fitness* local. Cet objectif de *fitness* local correspond à compter le nombre de voisins de plus petite *fitness* parmi les k plus proches.

NSLC est illustrée dans sa publication originale pour le problème de la vie artificielle, visant à faire évoluer des créatures virtuelles en trois dimensions de façon plus performante que *Novelty-Search* seule et une optimisation de la *fitness* seule. Elle est également utilisée pour apprendre à contrôler la marche d'un robot hexapode de façon omnidirectionnelle (Cully et Mouret, 2016). L'idée est de profiter de l'exploration de NSLC pour construire un répertoire de politiques de marche dans chaque direction simultanément, plutôt que d'apprendre les politiques séparément. Mouret (2011) utilise une méthode similaire qui combine *Novelty-Search* avec une optimisation multi-objectif pour conserver un objectif de performance et l'illustre pour la navigation d'un robot dans un labyrinthe.

Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) (Cully, Clune et al., 2015 ; Mouret et Clune, 2015) est un algorithme de qualité-diversité qui a pour origine la volonté d'illuminer un espace de solutions, c'est-à-dire de trouver dans un espace comportemental prédéfini et discrétisé en un nombre fini de cellules la meilleure solution de chaque cellule (micro-évolution), ce qui s'apparente à trouver l'élite de chaque niche écologique (macro-évolution) (Mouret, 2020).

MAP-Elites étant un algorithme qui sera utilisé et adapté à de nombreuses reprises dans la suite de cette thèse, nous allons le présenter en détail (Algorithme 1). L'idée principale est de former, comme NSLC, une archive de solutions avec plusieurs différences. D'une part, l'archive comportementale est prédécoupée sous forme de grille et ne nécessite pas de calcul sur les k plus proches voisins pour ajouter un élément. D'autre part, il n'y a pas de population. L'archive garde à jour dans chaque cellule la meilleure solution correspondant à ce comportement et en génère de nouvelles solutions à l'aide d'un opérateur de variation prenant en entrée une ou plusieurs élites de l'archive. Ceci permet, comparé à une population fluctuante, d'avoir une exploration globale de l'ensemble de l'espace des comportements à tout instant de l'exécution.

Une mesure de performance pour les problèmes de qualité-diversité est le QD-score (Pugh, Soros, Szerlip et al., 2015) qui consiste à calculer la moyenne de *fitness* de chaque élite de l'archive :

$$\text{QD-score} = \frac{1}{N} \sum_{\mathbf{b} \in A} \texttt{fitness}(A(\mathbf{b})),$$

où A est une archive de N élites, $\mathbf{b}$ est un descripteur de comportement et $A(\mathbf{b})$ l'élite dans l'archive A qui l'a généré. En attribuant une *fitness* nulle à une cellule vide, le QD-score permet de faire une moyenne entre la couverture (c'est-à-dire le pourcentage de cellules avec au moins une solution) et la qualité (c'est-à-dire les performances des solutions trouvées).

L'une des premières applications de MAP-Elites est la création d'un répertoire de marches pour un robot hexapode en simulation avec pour espace comportemental le ratio de temps de contact de chaque patte avec le sol (Cully, Clune et al., 2015). Ce répertoire est ensuite utilisé pour de l'adaptation aux dommages via optimisation bayésienne où les différentes élites sont

Algorithm 1

Paramètres :
X : l'espace des solutions
C : l'espace des comportements
B : le budget d'évaluation
N : le nombre total d'élites désirées
`fitness` : $\mathcal{X} \to \mathbb{R}$
`behavior` : $\mathcal{X} \to C$
Hyper-paramètres :
G : nombre de solutions aléatoires utilisées pour initialiser l'archive
`variation_operator` : $\mathcal{X} \to \mathcal{X}$: génère une nouvelle solution à partir d'une élite
Initialisation :
$A \leftarrow \emptyset$ ▷ Archive discrétisant C sous forme d'une grille multidimensionnelle de N cellules
Boucle Principale :
for $i \leq B$ **do**
  if $i \leq G$ **then**
    $\mathbf{x}' \leftarrow$ random_in(X) ▷ Initialize l'archive avec des solutions aléatoires
  else
    $\mathbf{x} \leftarrow$ random_in(A) ▷ Sélectionne une élite aléatoirement dans l'archive
    $\mathbf{x}' \leftarrow$ `variation_operator`($\mathbf{x}$) ▷ Genère une nouvelle solution
  $\mathbf{b} \leftarrow$ `behavior`($\mathbf{x}'$)
  if $A(\mathbf{b}) = \emptyset$ or `fitness`($A(\mathbf{b})$) < `fitness`($\mathbf{x}'$) **then**
    $A(\mathbf{b}) \leftarrow \mathbf{x}'$ ▷ Devient l'élite si la cellule est vide ou obtient une meilleure *fitness*
return A

utilisées pour permettre au même robot réel de marcher avec des dommages, par exemple une patte en moins. MAP-Elites s'illustre aussi dans de nombreux autres domaines, par exemple : la génération de scénarios d'évaluations pour de l'interaction homme-robot (Fontaine et Nikolaidis, 2020) ; la gestion et planification de main-d'œuvre (Urquhart et al., 2020) ; l'urbanisme (Galanos et al., 2021) ; le jeu vidéo avec de la découverte de stratégies différentes (Perez-Liebana et al., 2021), de la génération de contenu procédurale (Fontaine, R. Liu et al., 2021 ; Viana et al., 2022 ; Earle et al., 2022), et la conception innovante (Medina et al., 2023) ; le test de logiciel (Xiang et al., 2023) ; l'optimisation sous contrainte (Fioravanzo et al., 2021) ; et l'optimisation de portefeuille financier (Gašperov et al., 2024).

L'un des inconvénients de MAP-Elites est la création de l'archive à base d'une grille. En effet, une grille en haute dimension n'est pas viable car elle nécessite un nombre exponentiel de cellules pour couvrir avec la même résolution. CVT-MAP-Elites (Vassiliades, Chatzilygeroudis et al., 2017) propose d'utiliser une autre discrétisation via *centroidal Voronoi tessellation* (CVT). L'idée est de partitionner l'espace des comportements en N régions géométriques bien réparties, chaque région étant définie par un centroïde. Ces régions forment l'archive, de sorte qu'un nouveau comportement est rattaché à la cellule du centroïde le plus proche.

Le choix de l'opérateur de variation est important pour obtenir les meilleures performances. Les premières versions de MAP-Elites utilisaient le simple ajout d'un bruit gaussien (ce qui correspond uniquement à de la mutation). Il a ensuite été découvert que, dans le cas d'une archive d'élite, les variations à base de croisements permettent d'obtenir une bonne exploration d'un sous-espace des solutions, appelé l'hyper-volume des élites, qui augmente les chances que le nouvel individu soit aussi de haute performance (Vassiliades et Mouret, 2018). Les versions plus récentes utilisent donc *Simulated Binary Crossover* (SBX) (Agrawal et al., 2000), qui

prend deux individus et qui retranscrit le croisement de chaînes binaires pour des vecteurs réels. Plus formellement, étant donné deux parents x_1 et x_2, SBX génère un croisement comme suit :

$$\beta_q = \begin{cases} (2u)^{\frac{1}{\eta_c+1}} & \text{si } u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}} & \text{si } u > 0.5 \end{cases}$$

où $u \sim \mathcal{U}(0,1)$ et η_c est un paramètre d'intensité de croisement. Puis, il génère deux enfants :

$$c_1 = \frac{1}{2}\left[(1+\beta_q)x_1 + (1-\beta_q)x_2\right]$$
$$c_2 = \frac{1}{2}\left[(1-\beta_q)x_1 + (1+\beta_q)x_2\right]$$

Enfin,

$$\mathbf{x}_{\text{offspring}} = (x^i)_{1\leq i\leq n}, \text{ où } x^i = \begin{cases} c_1^i & \text{si } u^i \leq 0.5 \\ c_2^i & \text{si } u^i > 0.5 \end{cases}$$

où $u^i \sim \mathcal{U}(0,1)\ \forall 1 \leq i \leq n$.

Un nouvel opérateur plus simple que nous appellerons *iso-line* se base sur la même idée. Étant donné deux élites parents $\mathbf{x}_1$ et $\mathbf{x}_2$ l'opérateur de variation *iso-line* correspond à effectuer le calcul suivant :

$$\mathbf{x}_{\text{offspring}} = \mathbf{x}_1 + \sigma_{line}(\mathbf{x}_2 - \mathbf{x}_1)\mathcal{N}(0,1) + \sigma_{iso}\mathcal{N}(\mathbf{0}, \mathbf{I}),$$

où σ_{line} et σ_{iso} sont deux facteurs qui contrôlent l'intensité de croisement et de mutation. *Iso-line* peut aussi se voir comme une version simplifiée de SBX qui conserve une grande partie de ses performances.

2.5.2 Qualité-Diversité pour résoudre des problèmes d'apprentissage par renforcement profond

Ces opérateurs de variations sont assez simples et basés principalement sur une exploration stochastique, ce qui empêche MAP-Elites d'être applicable à des problèmes de haute dimension, en particulier l'optimisation de réseaux de neurones profonds. Cette section présente une série de travaux qui, pour la plupart, s'appuient sur MAP-Elites pour améliorer les performances et se rapprocher des capacités des méthodes d'apprentissage par renforcement profond.

2.5.2.1 Avec une stratégie d'évolution

Une première famille de méthodes exploite des stratégies d'évolution pour obtenir des opérateurs de variations plus performants.

MAP-Elites with Evolutionary Strategies (ME-ES) (Colas et al., 2020). ME-ES exploite les stratégies d'évolution pour permettre l'optimisation de réseaux de neurones profonds. De façon itérative, une des élites de l'archive est sélectionnée. Elle est optimisée à l'aide d'une stratégie d'évolution naturelle avec pour objectif : soit la *fitness*, soit la nouveauté, puis évaluée dans l'environnement, et ce, pour plusieurs itérations avant de sélectionner une nouvelle élite.

Dans sa publication originelle, ME-ES a été comparée à MAP-Elites pour construire un répertoire de marche pour l'environnement *Ant-v2* d'OpenAI Gym. MAP-Elites trouve beaucoup plus de comportements différents, mais de performances 3 à 4 fois inférieures. ME-ES a aussi été comparé dans deux tâches de pure exploration avec une récompense trompeuse et montre des performances similaires à l'état de l'art de l'exploration pure, NS-ES.

L'inconvénient de cette méthode est que l'optimisation via une stratégie d'évolution consomme une part importante du budget d'évaluation dans l'environnement, ce qui diminue le nombre total d'itérations de MAP-Elites.

Covariance Matrix Adaptation MAP-Elites (CMA-ME) (Fontaine, Togelius et al., 2020). CMA-ME est une variante de MAP-Elites qui incorpore CMA-ES pour améliorer les performances, en particulier le fait que le paramètre de variation de MAP-Elites est constant et pourrait bénéficier des capacités d'adaptation de CMA-ES. Pour ce faire, CMA-ME utilise différentes instances de CMA-ES appelées émetteurs, chacun mettant à jour sa propre distribution. Les auteurs définissent trois types d'émetteurs : (1) un émetteur aléatoire qui optimise aléatoirement une direction dans l'espace des comportements à chaque itération ; (2) un émetteur d'optimisation de la *fitness* qui utilise l'archive pour estimer la distribution (à la place de la population courante comme pour CMA-ES) ; et (3) un émetteur d'amélioration qui cherche, grâce à l'archive, les solutions qui ont de fortes chances de d'abord remplir des niches vides et ensuite d'améliorer les solutions des niches non vides. Chaque exécution de CMA-ME utilise 15 émetteurs du même type qui sont alternés de façon à ce que chacun génère le même nombre de solutions candidates.

Dans sa publication originelle, CMA-ME est d'abord comparée dans deux environnements jouets avec une forte déformation entre l'espace des solutions et l'espace des comportements dans le but de montrer l'utilité d'utiliser CMA-ES pour adapter le facteur de mutation. Les trois types de CMA-ME sont comparés à MAP-Elites original, CMA-ES, et MAP-Elites *iso-line*. CMA-ME avec l'émetteur aléatoire obtient le meilleur QD-Score en haute dimension car il remplit le plus de cellules. CMA-ME avec l'émetteur d'amélioration arrive second en remplissant un peu moins de cellules mais avec de meilleures solutions. C'est cette variante qui est choisie et par la suite appelée CMA-ME.

Toujours dans sa publication originelle, CMA-ME est ensuite comparée pour générer une politique via un réseau de neurones profonds pour jouer au jeu de cartes de duels *Hearthstone*. La *fitness* correspond à la différence de points de vie entre les deux adversaires à la fin de la partie. L'espace des comportements correspond au nombre moyen de cartes dans les mains et le nombre moyen de tours pour finir la partie. CMA-ME obtient de meilleures performances que l'ensemble des méthodes de référence et en particulier, CMA-ES qui converge rapidement vers une bonne *fitness* avant de se faire dépasser. Ceci montre encore qu'ajouter de la diversité comportementale dans des environnements complexes permet de trouver de meilleures solutions qu'un algorithme purement d'optimisation de performance.

CMA-ME s'illustre aussi pour la génération de scénarios de collaboration humain-robot (Fontaine, Hsu et al., 2021) ainsi que pour la génération de niveaux de jeux vidéo avec l'illumination d'espace latent (Fontaine, R. Liu et al., 2021) et d'automate cellulaire neuronal (Earle et al., 2022).

CMA-ME présente trois principales limitations : (1) l'émetteur d'amélioration a tendance à passer rapidement d'une région à une autre pour explorer de nouvelles régions alors que de meilleures solutions peuvent être trouvées dans la région actuelle (Tjanaka, Fontaine, Togelius

et al., 2022) ; (2) comme CMA-ES, CMA-ME ne fonctionne pas bien dans les environnements où la *fitness* est plate (Paolo et al., 2021) ; et (3) la résolution de l'archive a un fort impact sur les performances (Cully, 2021 ; Fontaine et Nikolaidis, 2021b).

Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) (Fontaine et Nikolaidis, 2023). CMA-MAE est une amélioration récente qui aborde ces limitations. Il introduit d'une part un paramètre pour transitionner en douceur entre CMA-ES et CMA-ME et, d'autre part, fournit une méthode simple pour adapter ce paramètre d'une résolution d'archive à une autre, résolvant ainsi la limitation (3). Ce paramètre permet de rendre l'archive "molle" dans le sens où le critère d'élitisme est moins strict. C'est-à-dire qu'en fonction d'un paramètre (s'apparentant à la température en recuit simulé), une solution moins bonne mais plus récente peut remplacer une meilleure solution plus ancienne. Ceci permet de rester plus longtemps dans des régions prometteuses et donc de résoudre la limitation (1). D'autre part, tant que ce paramètre est strictement entre 0 (CMA-ES) et 1 (CMA-ME), grâce au critère d'élitisme mou, CMA-MAE se comporte sur une région de *fitness* plate comme une descente de gradient de l'histogramme de l'archive, c'est-à-dire qu'il se déplace pour explorer les régions les moins explorées, répondant ainsi à la limitation (2).

Dans sa publication originelle, CMA-MAE est comparé à MAP-Elites original, MAP-Elites *iso-line*, et CMA-ME sur plusieurs jeux de référence de qualité-diversité et obtient à chaque fois de meilleurs QD-Score et couverture. Une amélioration (Tjanaka, Fontaine, D. H. Lee et al., 2023) de CMA-MAE lui permet de s'appliquer à des politiques de réseaux de neurones similaires à celles de l'apprentissage par renforcement profond et obtient ainsi les meilleures performances des variantes utilisant une stratégie d'évolution.

2.5.2.2 Avec un gradient de politique

Une seconde famille de méthodes exploite des méthodes de gradients de politique (comme TD3 (Fujimoto et al., 2018)) pour obtenir des opérateurs de variations qui permettent d'optimiser des politiques sous forme de réseaux de neurones profonds.

QD-RL (Cideron et al., 2020). QD-RL est une autre approche de qualité-diversité qui s'inspire de MAP-Elites pour l'utilisation d'une archive. QD-RL entraîne, via TD3, d'une part deux critiques, l'un pour la *fitness* et l'autre pour la diversité, et d'autre part, une population d'acteurs de qualité et de diversité. Le front de Pareto (l'ensemble des individus non strictement dominés par un autre) entre la qualité et la diversité est calculé via une optimisation multi-objective à partir de l'archive pour former à chaque génération la nouvelle population d'acteurs. Un inconvénient de cette méthode est le besoin de définir un comportement au niveau de chaque état s (pour entraîner le critique de diversité) alors que le comportement est plus couramment défini pour l'ensemble de la séquence d'états.

Dans sa publication originelle, QD-RL est comparé à ME-ES dans l'environnement *Ant-Maze* d'OpenAI Gym, où le but est d'apprendre à contrôler un agent quadrupède à se déplacer dans un labyrinthe (état de dimension 29 et action de dimension 8). QD-RL converge vers les mêmes performances en nécessitant 18 fois moins d'évaluations que ME-ES. Ceci peut s'expliquer par l'exploitation d'un tampon de mémoire qui permet, comme pour les méthodes d'apprentissage par renforcement profond hors ligne, d'être plus efficace en échantillonnage.

Policy gradient assisted MAP-Elites (PGA-ME) (Nilsson et al., 2021). PGA-ME préserve la structure globale de MAP-Elites à deux différences près. D'une part, chaque évaluation dans l'environnement est utilisée pour entraîner deux critiques via TD3 avec un

contrôleur glouton qui ne sert que pour leur entraînement et qui n'interagit pas avec l'environnement (ce qui est possible parce que TD3 est un algorithme d'apprentissage par renforcement profond hors ligne). D'autre part, en plus de l'opérateur de variation classique de MAP-Elites hérité des algorithmes d'évolution, PGA-ME utilise l'un des deux critiques appris pour optimiser les contrôleurs via plusieurs pas de descente de gradient. PGA-ME utilise ces deux opérateurs de manière équitable à chaque génération.

Dans sa publication originelle, PGA-ME est évalué sur un jeu de référence de QD sur OpenAI Gym contenant *QDWalker, QDHalfCheetah, QDAnt*, et *QDHopper*. La *fitness* correspond à la distance parcourue avec un malus de consommation d'énergie et les comportements correspondent à la proportion de contact de chaque membre avec le sol. PGA-ME est comparé à MAP-Elites, ME-ES, QD-RL, TD3 utilisant une archive et une ablation de PGA-ME avec uniquement le nouvel opérateur de variation. PGA-ME obtient une couverture identique ou supérieure à MAP-Elites et surtout parvient à trouver de meilleurs contrôleurs et obtient un meilleur QD-Score sur l'ensemble des tâches que toutes les autres méthodes.

MAP-Elites via a Gradient Arborescence (MEGA) (Fontaine et Nikolaidis, 2021b). MEGA est un algorithme de qualité-diversité qui s'intéresse aux problèmes différentiables, c'est-à-dire où la fonction de *fitness* et la fonction qui renvoie le comportement sont différentiables du premier ordre. MEGA se divise en deux variantes, *Objective and Measure Gradient MEGA* (OMG-MEGA) qui s'inspire de MAP-Elites et CMA-MEGA qui s'inspire de CMA-ME. OMG-MEGA consiste à doter MAP-Elites d'un nouvel opérateur de variation qui, au lieu de modifier la solution avec du bruit gaussien, réalise un pas de descente d'une arborescence de gradient. C'est-à-dire qu'il réalise un pas de gradient d'une fonction auxiliaire qui prend en compte la fonction de *fitness* mais aussi une direction aléatoire dans l'espace des comportements. CMA-MEGA correspond à la même chose mais optimise la direction dans l'espace des comportements par un émetteur d'amélioration de CMA-ES, c'est-à-dire qui cherche explicitement à améliorer le QD-Score.

Dans leur publication originelle, OMG-MEGA et CMA-MEGA sont comparés à MAP-Elites, MAP-Elites (*iso-line*), CMA-ME et une variante qui n'optimise que la fonction de *fitness*, OG-MAP-Elites. Les méthodes sont évaluées sur deux environnements de projection linéaire (sphère et Rastrigin), un bras articulé de 1 000 degrés de liberté et une tâche d'illumination d'espace latent de réseaux antagonistes génératifs d'images (*StyleGAN*) (Karras et al., 2020). CMA-MEGA obtient le meilleur QD-Score et la meilleure couverture sur chacun de ces problèmes.

CMA-MEGA peut aussi bénéficier des améliorations de CMA-MAE, pour devenir CMA-MAEGA qui obtient de meilleures performances que CMA-MEGA sur les mêmes problèmes.

2.5.2.3 Comparaison entre stratégie d'évolution et gradient de politique

Tjanaka, Fontaine, D. H. Lee et al. (2023) comparent les différentes méthodes sur les problèmes de QD-RL. Ils montrent que les variantes avec stratégies d'évolution peuvent obtenir des performances similaires aux méthodes basées sur un gradient de politique tout en nécessitant moins d'hyperparamètres (5 pour CMA-MAE et 6 pour ME-ES, contre 18 pour PGA-ME et 15 pour CMA-MEGA) et en ayant un meilleur temps de calcul en considérant le temps réel.

2.6 Alternatives à l'apprentissage par renforcement profond

L'apprentissage par renforcement profond se base sur le principe de guider un apprentissage autonome principalement via une fonction de récompense. Il existe bien d'autres paradigmes d'apprentissage qui, aujourd'hui, se trouvent dans l'ombre de l'apprentissage par renforcement profond. Cette section répertorie de telles méthodes qui présentent des alternatives viables à l'apprentissage par renforcement profond pour la recherche d'une politique.

2.6.1 Apprentissage d'un modèle inverse entre politiques et comportements

Une approche différente pour aborder le problème d'apprentissage de comportement est l'apprentissage de modèles inverses. Ceci correspond à modéliser l'environnement en deux espaces : l'espace des actions ou politiques que l'agent peut choisir à sa guise et l'espace des comportements qui résulte de l'action qu'effectue l'agent dans l'environnement. Cette représentation est une approche de recherche directe de politique (c'est-à-dire que la politique est vue comme un tout et qu'il n'y a pas de séquence d'actions). L'objectif étant de relier chaque comportement à une politique qui peut le générer. L'apprentissage par renforcement profond optimise un comportement sous forme d'une politique pour maximiser une récompense. À l'opposé, l'apprentissage de modèles inverses relie les comportements aux politiques qui les génèrent. De plus, selon la représentation choisie pour les comportements, la valeur de *fitness* peut être calculée a posteriori pour sélectionner le meilleur comportement trouvé et une politique qui le génère.

Une façon d'apprendre un tel modèle inverse est de s'inspirer des sciences cognitives et, plus particulièrement, de la façon dont les enfants en bas âge apprennent à interagir avec le monde (Oudeyer et al., 2007), non pas simplement en faisant des mouvements aléatoires, mais en cherchant à atteindre un but, comme par exemple saisir un objet particulier. Aubret et al. (2019) propose une revue détaillée de l'interaction des méthodes utilisant de la motivation intrinsèque avec l'apprentissage par renforcement profond. Ils identifient deux catégories de méthodes : les méthodes qui acquièrent des connaissances sur l'environnement et les méthodes qui apprennent des comportements. L'acquisition de connaissance s'intéresse à la dynamique de l'environnement avec l'exploration, la représentation de l'environnement ou la contrôlabilité (ce que l'agent peut réellement influencer dans l'environnement). La plus grande partie des méthodes s'intéresse à l'exploration en définissant des mesures telles que l'erreur de prédiction, la nouveauté ou le gain d'information. L'apprentissage de comportements se concentre principalement sur la création d'un curriculum d'apprentissage avec, par exemple, la notion de progrès d'apprentissage.

2.6.1.1 L'exploration dirigée par des buts aléatoires

Une approche naïve pour résoudre ce problème consiste à effectuer des actions aléatoirement et à garder en mémoire quelle action engendre quel comportement. Bien que facilement parallélisable, cette méthode a une très faible efficacité d'échantillonnage, surtout dans les environnements où l'espace des comportements n'est pas uniformément corrélé aux actions. Par exemple, dans le cas d'un robot qui essaie d'ouvrir une porte en tirant sur la poignée, la grande majorité des actions ne permettent pas d'atteindre la poignée et correspondent donc au même comportement d'immobilité de la poignée.

Les travaux Rolf et al. (2010) formalisent une solution plus avancée, qu'ils appellent l'exploration dirigée par des buts. Au lieu d'échantillonner aléatoirement dans l'espace des actions, l'idée est d'échantillonner un but dans l'espace des comportements puis d'utiliser une méthode de régression pour choisir une action qui a de fortes chances d'atteindre ce but. L'exploration dirigée par des buts permet d'apprendre à des robots dotés d'un nombre important de degrés de liberté. C'est-à-dire un robot humanoïde Honda avec 15 degrés de liberté pour sélectionner des postures de corps complet qui permettent d'atteindre avec sa main des points dans l'espace et jusqu'à 50 degrés de liberté pour un bras planaire atteignant des cibles.

Une technique simple pour atteindre un comportement but c_{but} est de regarder dans l'historique des paires (p, c) le comportement c' le plus proche de c_{but} puis de sélectionner la politique p' correspondante avec un bruit additionnel (le plus souvent gaussien). La politique p peut ainsi correspondre au paramètre d'une politique en boucle ouverte comme une DMP (Laversanne-Finot et al., 2021).

2.6.1.2 L'exploration dirigée par des buts intrinsèquement motivés

Les travaux Baranes et al. (2013) réemploient cette idée d'exploration dirigée par des buts mais, au lieu de choisir les buts aléatoirement, ils s'inspirent des sciences cognitives pour modéliser une motivation intrinsèque de l'agent, modélisant une curiosité (Oudeyer et al., 2007). Par exemple, dans le cadre d'un apprentissage, un élève va être plus intéressé à essayer de réaliser une tâche ni trop facile (qu'il maîtrise déjà) ni trop difficile (impossible ou pas encore accessible) au profit d'une tâche pour laquelle il peut progresser.

Ils proposent *Self-Adaptive Goal-Generation Robust Intelligent Adaptive Curiosity* (SAGG-RIAC), un algorithme d'apprentissage de modèle inverse de la dynamique. SAGG-RIAC découpe progressivement l'espace des tâches en sous-régions et calcule pour chacune une mesure d'intérêt qui correspond à la valeur absolue du progrès instantané (c'est-à-dire à quel point l'agent arrive mieux à atteindre les buts qu'il se fixe, calculé à partir d'une fenêtre glissante). Si l'agent s'améliore dans cette région, alors le progrès est positif et il peut être intéressant de continuer à y retourner. Si, au contraire, le progrès diminue (du fait d'une évolution de l'agent ou de l'environnement), cela indique que l'agent devient moins bon dans cette région et qu'il peut être intéressant d'y retourner. Et si le progrès stagne, c'est que soit l'agent a parfaitement appris à atteindre les buts qu'il se fixe dans cette région, soit que la région est trop difficile ou impossible. SAGG-RIAC utilise un modèle linéaire inverse pour sélectionner les actions en fonction des paires actions-comportement déjà rencontrées.

SAGG-RIAC permet d'accélérer l'exploration dans les environnements redondants avec un bras articulé de haute dimension, de sélectionner automatiquement un curriculum de tâches améliorant l'efficacité d'échantillonnage (pour des tâches de locomotion avec un robot quadrupède) et permet de découvrir les tâches impossibles à réaliser et donc de perdre moins d'échantillons à essayer de les résoudre (pour un bras articulé contrôlant une canne à pêche).

Intrinsically Motivated Goal Exploration Processes (IMGEP) (Forestier et al., 2022) s'inspirent des mêmes idées et généralisent le principe de but en tant que fonctions de *fitness* paramétrées. Ceci permet de regrouper à la fois la maximisation de récompense, le suivi de trajectoire et des contraintes d'égalité et d'inégalité. IMGEP permet d'apprendre plus efficacement un but particulier, la manipulation d'un objet, en choisissant activement différents sous-buts, par rapport à se focaliser uniquement sur ce but. Deux différences avec l'apprentissage par renforcement profond sont que d'une part, la fonction de *fitness* se calcule à la fin d'un épisode et n'est donc pas une récompense séquentielle, ce qui enlève le problème d'assi-

gnation de la récompense. D'autre part, elle se calcule pour n'importe quel but à partir de la trajectoire de l'agent, permettant ainsi de la réutiliser plusieurs fois au cours de l'apprentissage pour différents buts.

IMGEP remplit une archive de paires de politique et de comportement de façon similaire à MAP-Elites à la différence qu'il n'est pas élitiste et que toutes les paires sont conservées. Dans l'autre sens, MAP-Elites peut être vu comme une instanciation d'IMGEP où un but aléatoire est choisi parmi ceux déjà atteints (sélection d'un élite dans l'archive) et la politique choisie pour l'atteindre est une variation de la meilleure politique à l'avoir atteint (application de l'opérateur de variation sur l'élite). Une différence majeure est que pour IMGEP, la fonction de *fitness* est intrinsèquement générée par l'algorithme pour guider l'exploration. Les deux approches s'appuient tout de même sur deux grands principes : (1) l'utilisation d'une diversité de solutions globalement sous-optimales pour explorer et (2) l'utilisation d'une politique qui atteint un certain comportement avec une variation pour essayer d'atteindre un autre comportement.

Les idées développées par IMGEP forment l'un des axes de recherche pour améliorer l'exploration dans l'apprentissage par renforcement profond (Ladosz et al., 2022) et pour l'apprentissage continu tout au long de la vie (Parisi et al., 2019).

2.6.2 Apprentissage à partir de démonstrations humaines

Tout comme l'humain va à l'école pour apprendre, un autre paradigme d'apprentissage que l'exploration autonome via renforcement ou motivation intrinsèque est l'apprentissage par démonstration (Schaal, 1996b ; Ravichandar et al., 2020). Une autre façon d'entraîner un agent artificiel est donc d'exploiter l'ensemble des connaissances déjà acquises par l'humain.

Le premier exemple d'une telle idée est *Autonomous Land Vehicle In a Neural Network* (ALVINN) (Pomerleau, 1988), qui consiste à entraîner un réseau de neurones pour diriger un véhicule le long d'une route à partir de données créées en simulation par un expert. Schaal (1996a) étudient comment cette méthode peut s'appliquer à l'apprentissage par renforcement en détaillant l'impact sur la Q-fonction $Q(s, a)$, la fonction de valeur $V(s)$, la politique $\pi(a|s)$ et un modèle de transition $T(s'|s, a)$. Ils concluent sur l'intérêt théorique mais ne montrent des résultats positifs que pour l'apprentissage par renforcement basé sur un modèle, car les démonstrations permettent justement d'entraîner ce modèle plus efficacement qu'avec des données aléatoires.

Nécessitant un expert tout au long de l'apprentissage. Une méthode pour apprendre par démonstration est le clonage de comportements qui apprend directement une politique imitant les comportements des démonstrations. *Dataset Aggregation* (DAgger) (Ross et al., 2011) est un algorithme de clonage de comportements qui alterne entre : entraîner de façon supervisée une politique pour imiter les comportements (à partir d'un jeu de données de paires (s, a)), exécuter la politique pour collecter de nouvelles observations, demander à un expert quelles actions il aurait faites et ajouter ces nouvelles données au jeu de données. DAgger est évalué dans sa publication originelle sur un jeu de course en 3D (*Super Tux Kart*) et un jeu de plateformes (*Super Mario Bros.*), où il performe plus efficacement qu'une méthode purement supervisée et une autre méthode de clonage de comportement itératif.

DAgger a aussi été utilisé pour apprendre à voler à un drone dans des tuyaux où les perturbations dues aux turbulences rendent le contrôle difficile (Tempez, 2022). Plus précisément, une version modifiée de DAgger est utilisée pour entraîner une politique "légère" via un réseau de neurones pour pouvoir être exécutée à bord du drone en imitant comme expert un contrôle en commande prédictive. La modification par rapport à l'algorithme original est l'agrégation

dans le jeu de données d'apprentissage supervisé non pas uniquement de la première commande prédite par la commande prédictive mais de toute la séquence de commandes. Sans cette modification, la politique apprise ne permet pas une stabilisation suffisante du vol.

HG-DAgger (Kelly et al., 2019), une variante de DAgger, permet à l'expert humain d'intervenir pour respecter des contraintes de sûreté et démontre son importance pour des problèmes dangereux comme la conduite autonome. Une forte limitation de DAgger reste qu'il nécessite l'intervention d'un expert au cours de l'apprentissage et ne peut donc pas s'appliquer avec juste un jeu de données de démonstrations.

À partir d'un jeu de données de démonstrations. D'autres méthodes ne nécessitent l'expert que pour générer un ensemble de données de démonstrations.

Une telle méthode est par exemple le renforcement inverse, qui consiste à apprendre une fonction de coût qui favorise les trajectoires proches des démonstrations. *Generative Adversarial Imitation Learning* (GAIL) (Ho et al., 2016) est un algorithme s'inspirant de ce principe mais qui montre, en étudiant la formulation duale de l'apprentissage par renforcement et de l'apprentissage par renforcement inverse, qu'il est possible de directement apprendre une politique qui imite les démonstrations. De plus, GAIL se rapproche de la formulation des réseaux antagonistes génératifs (GAN) (Goodfellow et al., 2014), en utilisant une descente de gradient avec Adam (Kingma et al., 2014) pour entraîner un critique et une étape de TRPO (Schulman, Levine et al., 2015) pour entraîner un acteur. GAIL est évalué dans sa publication originale sur une suite d'environnements d'OpenAI Gym et est efficace d'un point de vue de l'utilisation des données des experts mais requiert le même ordre de grandeur d'échantillons d'interaction avec l'environnement que TRPO. Infogail (Y. Li et al., 2017) modifie GAIL pour obtenir une représentation latente des démonstrations de l'expert qui permet d'obtenir des informations d'explicabilité.

Une autre méthode plus récente est le clonage de comportement implicite (Florence et al., 2022) qui cherche à remplacer l'apprentissage d'une politique qui, pour un état s en entrée, renvoie une action optimale a^* par l'optimisation d'une fonction d'énergie $E(s, a)$ telle que

$$a^* = \underset{a}{\operatorname{argmin}} E(s, a).$$

Cette formulation en modèles à base d'énergie (LeCun, Chopra et al., 2006) apporte de nombreux avantages : l'apprentissage de fonctions d'énergie discontinues, l'extrapolation des données hors de l'enveloppe convexe des exemples, la possibilité d'apprendre des fonctions à plusieurs valeurs et une meilleure généralisation à partir de données visuelles (comparée à un réseau convolutionnel entraîné avec une erreur quadratique moyenne). Cette méthode obtient de meilleures performances que l'équivalent explicite pour un jeu de données d'évaluation d'apprentissage par renforcement hors ligne D4RL (J. Fu et al., 2020), tout en obtenant des performances similaires à l'état de l'art de l'apprentissage par renforcement profond hors ligne. Elle obtient aussi de meilleures performances que les méthodes explicites sur plusieurs tâches de manipulation avec un bras robotique xArm6, y compris sur le robot réel. Le clonage de comportement implicite est également utilisé pour des robots humanoïdes (Tanaka et al., 2023) et la conduite autonome (X. Wang et al., 2024).

À partir uniquement d'observations. *Behavioral Cloning from Observation* (BCO) (Torabi et al., 2018) est un algorithme d'apprentissage par démonstration qui a pour objectif supplémentaire d'apprendre avec uniquement un accès aux observations des interactions avec l'environnement. Ceci peut avoir un intérêt car créer un jeu de données de démonstration

nécessite d'avoir accès aux données internes à l'environnement et à l'agent. En comparaison, l'internet regorge d'exemples, en particulier de vidéos, pour un très large type de contenu et le plus souvent, ces vidéos ne contiennent pas d'information sur les actions prises.

BCO apprend d'abord de façon auto-supervisée un modèle inverse de la dynamique de l'environnement $P(a_t|s_t, s_{t+1})$. Il reçoit ensuite une démonstration sous forme de séquence d'états et utilise son modèle appris pour inférer une séquence d'actions en utilisant une méthode de maximisation de vraisemblance. Une seconde version de BCO est aussi présentée. De façon itérative, la trajectoire ainsi calculée est utilisée pour collecter de nouvelles données afin d'améliorer le modèle inverse de la dynamique pour améliorer le calcul de la trajectoire jusqu'à ce qu'un budget d'interactions avec l'environnement soit atteint.

BCO est évalué dans sa publication originelle sur plusieurs environnements classiques d'évaluation de l'apprentissage par renforcement profond d'OpenAI (Brockman et al., 2016) (*CartPole*, *MountainCar*, *Reacher* et *Ant*). BCO montre des résultats similaires (un peu moins bons selon l'environnement) que les méthodes de l'état de l'art d'apprentissage par renforcement (incluant GAIL) qui ont accès aux actions. Par contre, il nécessite plusieurs ordres de grandeur d'interactions en moins avec l'environnement pour imiter la démonstration (en comptant le total pour apprendre le modèle de la dynamique inverse et pour imiter la démonstration). Betz et al. (2021) propose aussi une amélioration qui utilise un mécanisme d'attention pour éviter les solutions sous-optimales.

L'apprentissage par démonstration n'est pas une méthode récente mais connaît ces dernières années un regain d'intérêt (Ravichandar et al., 2020). Par exemple, des travaux récents remplacent l'apprentissage par renforcement par de l'apprentissage par démonstration pour apprendre des comportements de manipulation avec des bras robotiques via des politiques de diffusion (Chi et al., 2023) ou des *transformers* (T. Z. Zhao et al., 2023 ; Z. Fu et al., 2024). L'apprentissage par démonstration permet de résoudre des problèmes où la définition d'un comportement optimal (en contrôle classique) ou d'une fonction de récompense pour guider l'optimisation (en apprentissage par renforcement) n'est pas aisée. Les deux principales limitations restent la nécessité d'avoir un expert humain pour générer des démonstrations et la faible généralisation à de nouveaux comportements.

2.6.3 Génération automatique de solutions pour l'apprentissage

Dans la continuité de l'apprentissage par démonstration via un expert, d'autres méthodes apprennent des politiques via un processus de génération automatique de démonstrations.

Utiliser une méthode classique comme expert. Un des premiers exemples est *Guided Policy Search* (GPS) (Levine et Koltun, 2013). GPS est un algorithme de recherche directe de politique datant d'avant l'essor de l'apprentissage profond. GPS utilise une méthode de programmation différentielle de la dynamique (DDP) (Jacobson et al., 1970) pour générer des trajectoires maximisant la récompense. Ces trajectoires sont ensuite utilisées par un échantillonnage préférentiel pour optimiser la politique à l'aide d'une méthode quasi-Newton du second ordre. L'échantillonnage préférentiel permet en effet de faire de l'apprentissage hors ligne et d'utiliser des échantillons provenant d'une autre politique que celle optimisée. Ceci permet, en initialisant le DDP avec des trajectoires de démonstrations, de réaliser de l'apprentissage de politique par démonstration. Dans la publication originale, GPS permet ainsi de contrôler en simulation dans un plan en deux dimensions un Marcheur, Nageur, et Sauteur, ainsi qu'un modèle en trois dimensions d'un humanoïde pour de la course sur du terrain non plat. GPS a fait l'objet de nombreuses améliorations (J. Du et al., 2021), permettant son

illustration dans plusieurs domaines de la robotique, par exemple la marche bipède (Levine et Koltun, 2013 ; Levine et Koltun, 2014), la conduite de drone autonome (T. Zhang et al., 2016) et la manipulation visio-motrice (Levine, Finn et al., 2016 ; Chebotar, Kalakrishnan et al., 2017 ; Chebotar, Hausman et al., 2017).

Créer un répertoire de primitives. *Evolutionary Repertoire-based Control* (EvoRBC) (Duarte et al., 2016 ; Duarte et al., 2018) est une méthode d'apprentissage qui découpe le problème de navigation dans un labyrinthe en (1) une étape d'évolution d'un répertoire de primitives de locomotion et (2) l'évolution d'un contrôleur de haut niveau qui exploite les primitives apprises.

Pour réaliser l'étape (1), EvoRBC utilise MAP-Elites pour construire un répertoire de primitives permettant au robot de se déplacer dans différentes directions (par exemple, des primitives pour avancer, reculer, aller à gauche, et aller à droite).

L'idée de l'étape (2) est de décomposer n'importe quelle trajectoire en une séquence de primitives. Par exemple, pour aller en avant et à droite, il suffit d'abord d'avancer puis d'aller à droite. Pour ce faire, EvoRBC définit une fonction d'indexation de chaque primitive et évolue un réseau de neurones qui prend en entrée les données sensorielles du robot et renvoie l'index d'une des primitives. Ensuite, pour une tâche donnée, par exemple une tâche d'exploration dans un labyrinthe, un contrôleur de haut niveau est évolué à l'aide de l'algorithme de neuro-évolution NEAT.

Dans sa publication originelle, EvoRBC permet à un robot à roues et à un robot hexapode en simulation de résoudre différents problèmes de navigation avec un score maximal aussi bon que celui de l'état de l'art mais en étant moins sensible à la complexité de la locomotion (grâce à la pré-construction du répertoire de primitives qui permet de séparer l'apprentissage du contrôle du robot de l'apprentissage de la résolution du problème de navigation). Une idée complémentaire à EvoRBC est l'utilisation d'une archive hiérarchique pour permettre la complexification des comportements en permettant d'utiliser les primitives pour apprendre de nouveaux comportements et ce, sur plusieurs niveaux de complexification (Cully et Demiris, 2018).

Séparer explicitement l'exploration et la résolution de l'apprentissage d'une politique robuste et générale. L'un des points faibles les plus handicapants des algorithmes d'apprentissage par renforcement profond est leur faible capacité à explorer dans des environnements à récompenses éparses et/ou contenant de nombreux minima locaux. Dans Ecoffet et al. (2021), les auteurs identifient deux problèmes distincts qui empêchent ces méthodes d'explorer : (1) ce qu'ils appellent le *détachement*, correspondant au fait que l'algorithme n'a pas de mécanisme lui permettant de revenir dans des états intéressants déjà visités et (2) le *déraillement*, qui correspond au fait que, même si l'algorithme se donne pour objectif de visiter un tel état, la politique courante a le plus souvent du mal à revenir précisément à cet état. Par exemple, dans une tâche d'exploration dans un labyrinthe, il peut être intéressant de s'éloigner de la sortie pour prendre un passage, mais un tel état amène à une décroissance de la récompense et donc à un manque d'intérêt pour y revenir (*détachement*). De plus, même si l'algorithme se donne pour objectif d'y retourner, si celui-ci est doté d'un mécanisme d'exploration trop puissant (par exemple avec un bruit gaussien élevé), il aura du mal à suivre le parcours exact qui amène à cet état pivot (*déraillement*).

Les auteurs proposent *Go-Explore*, un algorithme de résolution de problèmes d'apprentissage par renforcement, qui résout explicitement les deux problèmes cités ci-dessus. Pour ce faire, il propose de construire une archive des états parcourus associée à un score d'intérêt (pour éviter le problème de *détachement*) et propose comme mécanisme d'exploration de séparer le retour à l'un de ces états de l'étape d'exploration "libre".

Plus formellement, *Go-Explore* :

1. crée une archive d'états, en stockant pour chacun le nombre de visites et la récompense correspondante, ce qui permet de calculer, avec une heuristique, un score d'intérêt pour l'exploration ;
2. choisit l'un de ces états en favorisant le score d'intérêt puis y retourne soit directement si l'environnement le permet (ce qui est possible dans la plupart des simulateurs et moteurs de jeux vidéo) ou à l'aide d'une politique spécifiquement dédiée à retourner à un état but ;
3. explore à partir de cet état comme le ferait une méthode classique pour obtenir un état final ;
4. met à jour l'archive soit en ajoutant cet état final s'il est nouveau, soit en mettant à jour ses informations d'intérêt ;
5. retourne à l'étape 2 jusqu'à obtention d'une exploration suffisante de l'environnement ;
6. entraîne par démonstration une politique robuste comme étape finale.

Nous pouvons y voir deux hiérarchies : d'une part, *Go-Explore* sépare les épisodes en deux étapes : retour à un état intéressant puis exploration, ce qui permet de ne pas recommencer à zéro chaque exploration. Et d'autre part, il sépare la phase d'exploration de la phase d'apprentissage d'une politique robuste. La phase d'exploration est inspirée de la construction d'une archive de *Novelty-Search*, MAP-Elites, ou IMGEP.

Dans sa publication originelle, ceci permet à *Go-Explore* d'obtenir de meilleures performances que l'état de l'art sur le jeu de problèmes de référence composé des jeux Atari et de dépasser pour la première fois le record du monde (détenu jusqu'alors par un humain). *Go-Explore* permet aussi à un bras robotique (en simulation) d'apprendre à " ramasser puis déposer" des objets avec un taux de réussite de 99%, là où PPO n'y arrive pas du tout (c'est-à-dire qu'il ne voit aucune récompense positive tout au long de l'entraînement).

Un autre point faible des méthodes d'apprentissage par renforcement profond est leur faible efficacité d'échantillonnage. En effet, elles nécessitent souvent plusieurs millions d'évaluations avec l'environnement pour s'entraîner. Dans Norman et al. (2023), les auteurs proposent trois explications. Premièrement, chaque apprentissage débute de zéro, c'est-à-dire que les méthodes d'apprentissage par renforcement n'utilisent pas de connaissance a priori pour biaiser l'apprentissage comme le ferait un humain qui sait déjà beaucoup de choses de ses interactions passées. Deuxièmement, l'exploration est majoritairement conditionnée par le renforcement, c'est-à-dire à améliorer les mêmes comportements pour maximiser la récompense, plutôt que de maximiser l'exploration ou l'exploitation d'une découverte intéressante. Par exemple, dans un jeu d'exploration, un humain qui découvre une clé à un endroit va, à l'épisode suivant, chercher la porte que cette clé ouvre. D'autre part, cela engendre une exploration redondante, le même comportement peut être répété en boucle dans l'espoir d'améliorer la récompense, sans mécanisme particulier pour alterner différents comportements. Troisièmement, la même politique est utilisée pour explorer (collecter de nouvelles données pour améliorer la politique) et pour exploiter (maximiser la récompense).

Les auteurs proposent *First-Explore, then Exploit*, un algorithme de méta-apprentissage par renforcement qui essaie de résoudre ces problèmes en utilisant deux politiques, l'une pour l'exploration et l'autre pour l'exploitation. Les deux politiques sont conditionnées par un contexte (l'ensemble des épisodes antérieurs) et prennent la forme d'un *transformer* (GPT-2 (Radford et al., 2019)) permettant la mise en entrée d'une séquence de données. Elles partagent les mêmes poids sauf pour la dernière couche.

Une première limitation est le potentiel danger de la phase d'exploration. En effet, cette politique ignore la récompense dans laquelle peut se trouver un terme qui correspond à des contraintes de sécurité, et donc ne prend pas en compte la sécurité. Une deuxième limitation est que la politique d'exploration est optimisée de façon gloutonne, c'est-à-dire de façon à maximiser l'exploration de l'épisode en cours et non pas de l'ensemble des explorations futures. Ceci pose un problème de sous-optimalité de l'exploration. Une troisième limitation est le coût en calcul pour faire fonctionner des *transformers*.

2.6.4 D'abord qualité-diversité, puis généralisation

Les algorithmes de qualité-diversité génèrent pour un problème donné un ensemble de solutions diverses et de haute qualité sous forme d'une archive en mêlant recherche de diversité comportementale et optimisation. Ce riche jeu de données semble être une source idéale pour entraîner un modèle à généraliser ces solutions. Cette section présente des méthodes qui exploitent cette idée.

2.6.4.1 Apprentissage d'une représentation des "bonnes" solutions

Une première façon d'exploiter ce jeu de données est d'apprendre une représentation de ces solutions. L'idée est de pouvoir ensuite exploiter cette représentation pour explorer l'espace des solutions de haute qualité. *Data-Driven Encoding MAP-Elites* (DDE-ME) (Gaier et al., 2020) est une adaptation de MAP-Elites qui apprend une "bonne" représentation de l'espace des solutions grâce à un auto-encodeur variationnel, ce qui permet une recherche plus rapide de bonnes solutions. L'idée est que les solutions performantes existent dans un sous-espace de l'espace des solutions et qu'apprendre une représentation qui focalise la recherche dans ce sous-espace permet de concentrer l'exploration sur ces bonnes solutions (Vassiliades et Mouret, 2018). En alternant l'apprentissage de cette représentation via une archive d'élites et l'amélioration de l'archive d'élites à l'aide de cette représentation, la représentation converge vers l'hyper-volume des élites.

Plus formellement, DDE-ME alterne trois phases. (1) L'auto-encodeur variationnel est entraîné de façon non supervisée sur les élites de l'archive (initialisée avec des solutions aléatoires) afin d'apprendre un espace latent par reconstruction des élites. (2) L'algorithme de bandit UCB1 (Auer et al., 2002) optimise le choix entre plusieurs opérateurs de variation : mutation gaussienne isométrique, *iso-line* et un nouvel opérateur qui renvoie la moyenne entre une élite et sa reconstruction. Cet opérateur exploite l'auto-encodeur variationnel en projetant vers le sous-espace des élites une solution, ce qui permet d'étendre cet espace. (3) Les nouvelles solutions sont ajoutées dans l'archive, améliorant ainsi les solutions.

Dans sa publication originelle, DDE-ME est comparé à des ablations qui utilisent différentes combinaisons d'opérateurs de variation. Ces méthodes sont évaluées sur un environnement de bras articulé en deux dimensions doté de 20, 200 ou 1 000 degrés de liberté. La solution correspond aux positions articulaires, le comportement à la position de l'organe terminal en deux dimensions et la *fitness* à la variance négative des articulations. Dans une première

expérience d'illumination de l'espace des comportements, DDE-ME permet de meilleures QD-Scores et couverture. Dans une seconde expérience, la représentation apprise (en 10D et 32D) est utilisée pour remplacer la représentation directe des solutions (en 20D, 200D et 1 000D), ce qui permet à MAP-Elites de reconstruire l'archive avec plusieurs ordres de grandeur d'évaluations en moins qu'avec la représentation directe. Dans une troisième expérience, la représentation apprise est utilisée par CMA-ES pour réaliser de la cinématique inverse, ce qui lui permet d'être encore au moins un ordre de grandeur plus précis tout en conservant des solutions avec une faible variance entre les différentes articulations. Cela montre que le savoir acquis par MAP-Elites peut être distillé dans une représentation et non directement dans une politique, sachant qu'une méthode d'optimisation comme CMA-ES peut utiliser cette représentation pour générer une politique.

Rakicevic et al. (2021) ré-exploitent le même procédé pour concevoir un opérateur de variation qui réalise des perturbations au niveau de l'espace latent appris par l'auto-encodeur variationnel à l'aide de la jacobienne du décodeur. L'utilisation d'un espace latent n'est pas aussi immédiate qu'une politique mais permet peut-être de conserver une plus grande diversité.

2.6.4.2 Distillation des solutions dans une politique

Une autre façon d'exploiter le jeu de données généré par une méthode de qualité-diversité est de la distiller dans une politique.

Apprentissage d'un générateur de politiques conditionnées par un but. Jegorova et al. (2020) proposent une telle méthode en utilisant d'abord NSLC pour construire un répertoire de comportements, puis de distiller ces comportements à l'aide de réseaux antagonistes génératifs de politique (GPN) conditionnés par un but. Dans sa publication originelle, cette méthode est évaluée dans une tâche de lancer de balle avec le robot de manipulation Baxter dans un environnement contenant des obstacles. Le but étant de trouver différents comportements pour lancer la balle sur une cible afin qu'à l'inférence, si un comportement ne fonctionne pas à cause d'un obstacle, le robot ait accès à d'autres comportements pour essayer d'éviter les obstacles.

Sans obstacle, comparé à réutiliser directement les comportements du répertoire, le GPN permet d'échantillonner des comportements de moins bonne qualité (en termes de précision pour atteindre la cible) mais avec une plus grande diversité (en termes de trajectoire du bras du robot et de la balle). De plus, la diversité et la qualité du GPN se généralisent bien au robot réel.

Avec des obstacles, le GPN permet de trouver un comportement viable dans plus de 50% des cas alors qu'utiliser le répertoire seul ne permet de le faire que dans 30% des cas. Le GPN est aussi comparé avec une méthode d'estimation par noyau (KDE) utilisant les mêmes données pour inférer de nouveaux comportements et l'optimisation bayésienne avec comme a priori le meilleur comportement du répertoire et 10 essais pour l'optimiser. Les résultats montrent que l'utilisation d'un GPN permet encore une plus grande diversité comportementale et donc de résoudre plus de lancers de balle avec obstacles.

Une limitation de cette méthode est qu'elle ne permet pas de conditionner directement sur le comportement, ce qui nécessite de la relancer jusqu'à obtenir le comportement souhaité. Une alternative est donc de conditionner non pas sur un but mais directement sur le comportement.

Apprentissage d'une politique conditionnée par le comportement. *Descriptor-Conditioned Gradients MAP-Elites* (DCG-ME) (Faldor et al., 2023) est un algorithme de qualité-diversité qui peut s'appliquer à l'évolution de réseaux de neurones et qui fournit "gratuitement" une politique capable d'exécuter l'ensemble des solutions de l'archive. Plus formellement, DCG-ME apporte deux modifications à PGA-ME.

Premièrement, le critique est conditionné par les descripteurs de diversité comportementale. En effet, un problème dans l'utilisation de PGA-ME est que le critique optimise uniquement la *fitness*. Ceci pose problème si la solution optimale au problème est unique car dans ce cas, l'opérateur de variation ne ferait que muter chaque solution vers cette même solution unique, diminuant ainsi la diversité des solutions. En conditionnant le critique par le comportement, l'opérateur de variation conserve l'objectif de diversité. Dans sa publication originelle, ceci permet à DCG-ME d'améliorer le score de qualité-diversité de PGA-ME de 82% en moyenne sur un jeu de problèmes (*Ant-Omni*, *AntTrap-Omni*, *hexapode-Omni* et *Walker-Uni*) de référence en qualité-diversité QDGym inclus dans OpenAI Gym.

Deuxièmement, l'utilisation de TD3 implique l'apprentissage d'un acteur en parallèle du critique. DCG-ME met en avant cet acteur, tout aussi conditionné par le comportement, comme un produit de l'algorithme. Cet acteur est donc similaire au GPN présenté précédemment, à la différence que l'acteur est conditionné par le comportement et non pas juste un objectif. Ceci permet d'obtenir directement le comportement souhaité sans avoir à réechantillonner. Cette politique est évaluée sur l'ensemble des tâches et montre sur les tâches les plus simples (*Ant-Omni* et *AntTrap-Omni*) une bonne performance à reproduire l'archive mais n'y arrive pas sur les tâches les plus complexes (*hexapode-Omni* et *Walker-Uni*) du fait des actions de plus grande dimension.

Des travaux récents (Hegde et al., 2024) utilisent une idée similaire à l'aide de modèles de diffusion (L. Yang et al., 2023) pour compresser les politiques. Sur un problème de contrôle d'humanoïde en simulation, ils parviennent à compresser la taille avec un taux de 13 tout en conservant 98% de la *fitness* et 89% de la couverture.

Apprentissage d'une politique robuste. Macé et al. (2023) proposent une modification des algorithmes de qualité-diversité pour optimiser explicitement la robustesse des solutions trouvées. Ils proposent une variante de MAP-Elites, MAP-Elites Low-Spread (ME-LS), ainsi qu'une variante de PGA-ME, PGA-ME-LS. Ils entraînent ensuite un *transformer* conditionné par les comportements pour pouvoir les reproduire. La raison est qu'utiliser un réseau de neurones acteur pour représenter une politique pose un problème de robustesse à reproduire le même comportement dans des états initiaux légèrement différents (Flageat et al., 2023). En comparaison, un *transformer* permet de mieux prendre en compte la structure séquentielle des comportements. Plus formellement, l'apprentissage se déroule en trois étapes.

Premièrement, ME-LS construit une archive de comportements divers en cherchant explicitement à promouvoir les comportements robustes. Pour ce faire, ME-LS utilise une mesure de la dispersion d'une solution, c'est-à-dire sa tendance à engendrer des comportements différents si placée dans des états initiaux différents. La modification par rapport à MAP-Elites est le changement de la condition pour devenir l'élite qui, au lieu de juste avoir une *fitness* supérieure à l'élite actuelle, devient la condition d'avoir à la fois une meilleure *fitness* et une plus faible dispersion. La contrepartie est de devoir évaluer plusieurs fois chaque solution candidate.

Deuxièmement, l'algorithme génère un jeu de données d'entraînement en subdivisant l'espace des comportements en sous-zones et en sélectionnant pour chacune la solution de l'archive qui obtient le moins de dispersion dans cette zone. Chaque solution est ensuite exécutée pour obtenir un jeu de données de séquence de triplets de descripteur de comportement, d'action et d'état (b, s, a).

Troisièmement, un *Quality-Diversity Transformer* (QDT) est entraîné de façon supervisée sur ce jeu de données. QDT a comme architecture celle de GPT-2. Ce QDT est entraîné à prédire ces séquences d'actions de façon à ce que, durant l'inférence à partir d'un état initial et d'un descripteur, il puisse générer une séquence d'actions qui atteint ce comportement. Les QDT provenant de données de ME-LS et PGA-ME-LS permettent d'obtenir un meilleur score de robustesse que les archives d'où proviennent les données. Ils montrent aussi lors d'une évaluation de la généralisation à partir d'un jeu de données tronqué une capacité à interpoler les solutions pour retrouver l'ensemble des comportements.

Optimisation multi-tâche puis apprentissage d'une politique générale. *Evolved Environmental Trajectory Generators* (EETG) (Surana et al., 2023) sépare l'apprentissage de la marche pour un robot quadrupède entre une première phase de découverte de générateurs de marche sur des terrains variés via de la qualité-diversité et une seconde phase de distillation de ces marches en une seule politique.

Plus formellement, la première phase utilise une variante de Multi-Task MAP-Elites (MT-ME) (Mouret et Maguire, 2020) pour découvrir un ensemble de générateurs de marche pour une diversité d'environnements. Chaque générateur de marche correspond à une trajectoire périodique de chaque patte en boucle ouverte. Les différents environnements définissant l'archive de MT-ME sont définis à la main comme 80 environnements différents faisant varier quatre types d'environnements (des pentes, des escaliers, des terrains irréguliers ou des poteaux d'équilibre) avec différentes intensités. EETG exécute MT-ME, c'est-à-dire qu'il initialise l'archive avec des générateurs de trajectoires aléatoires, puis sélectionne un élite, applique un opérateur de variation, le teste dans un des environnements, et met à jour l'archive en fonction de la *fitness* obtenue par ce nouvel individu, pour un total d'un peu moins de 400 000 évaluations.

La seconde phase s'inspire de *Policies Modulating Trajectory Generators* (PMTG) (Iscen et al., 2018) pour apprendre une unique politique sous forme d'un réseau de neurones profond conditionné par les paramètres du générateur de trajectoire. La politique est entraînée via *Augmented Random Search* (ARS) (Mania et al., 2018). À chaque itération, un des environnements est aléatoirement sélectionné avec le générateur de trajectoire élite correspondant dans l'archive. La politique prend en entrée l'état du robot, les paramètres du générateur de trajectoire, et la fréquence de marche de chaque patte, et renvoie (1) des paramètres de phase pour moduler le générateur de trajectoire et (2) des résiduels additionnés à la position de chaque pied. Ces positions sont envoyées à un contrôleur bas niveau qui utilise la cinématique inverse et un contrôleur PID pour contrôler le robot.

Dans sa publication originelle, EETG est comparé à PMTG, qui optimise la politique avec ARS à partir d'un seul générateur de trajectoire prédéfini, et à CMA-ES, qui optimise un seul générateur de trajectoire avant d'optimiser la politique avec ARS. Par équité, EETG est comparé à deux variantes de chaque méthode, une méthode qui apprend une seule politique conditionnée par les paramètres de l'environnement et une méthode qui apprend une politique experte pour chaque environnement (ce qui multiplie grandement le nombre d'évaluations nécessaires pour l'entraînement). Chaque politique est évaluée sur l'ensemble des 80 environnements, le score final correspondant à la moyenne avec 20 réplications. EETG obtient

les meilleures performances bien que nécessitant cinq fois moins d'échantillons que les experts entraînés via CMA-ES, ce qui peut s'expliquer par plusieurs raisons : (1) un cercle vertueux d'apprentissage sur les différents environnements, l'un pouvant servir à améliorer les performances d'un autre ; (2) une meilleure utilisation des données en se focalisant sur une seule politique conditionnée par l'environnement ; et (3) en profitant de la diversité de générateurs de trajectoires optimisés lors de la première phase de qualité-diversité. L'un des inconvénients de cette méthode est la supposition de la connaissance des paramètres de l'environnement.

La distillation d'une archive créée par qualité-diversité semble une idée prometteuse, comme tendent à le montrer ces travaux récents (Jegorova et al., 2020 ; Faldor et al., 2023 ; Macé et al., 2023 ; Surana et al., 2023). C'est en tout cas cette méthodologie que nous emprunterons dans cette thèse.

2.7 Conclusion

L'apprentissage par renforcement profond est une méthode d'apprentissage complexe mais qui permet l'apprentissage de comportements par la seule interaction avec l'environnement. Bien qu'il ne nécessite pas de démonstrations humaines, l'optimisation des comportements est tout de même guidée par une fonction de récompense définie par un expert. Cette restriction des biais d'apprentissage rend les méthodes d'apprentissage par renforcement profond peu efficaces en échantillonnage. Il existe toutefois différentes méthodes pour l'améliorer, telles que l'apprentissage hors ligne qui réutilise des données, l'apprentissage basé sur un modèle qui utilise les données pour apprendre un modèle et obtenir des estimations d'échantillons "gratuits", et le méta-apprentissage qui factorise l'apprentissage de plusieurs tâches. L'apprentissage par renforcement profond n'est d'ailleurs pas la seule méthode pour entraîner une politique sous forme d'un réseau de neurones profond. Les méthodes de stratégies d'évolution utilisées en neuro-évolution sont compétitives et plus simples à mettre en place, même si elles conservent le défaut d'être tout autant peu efficaces en données.

L'apprentissage est guidé principalement par la fonction de récompense, même si le choix des représentations de l'espace des actions et des états et de l'architecture peuvent être considérés comme des biais d'apprentissage. Par exemple, un réseau de neurones convolutionnel est adapté aux problèmes de vision et les *transformers* aux problèmes séquentiels.

Le fait de se restreindre à peu de biais d'apprentissage pose problème à deux niveaux. Au niveau du paradigme d'apprentissage, se forcer à réapprendre tout pour résoudre une nouvelle tâche nécessite un effort important. Il est pertinent dans le cas d'étude de l'apprentissage en lui-même mais est un frein important dans le contexte de doter un agent artificiel de comportements. Si le but est uniquement d'obtenir un comportement, il est plus pertinent de profiter des connaissances existantes pour accélérer l'apprentissage, par exemple via démonstration. Dans le cas de la robotique, la dynamique est assez bien modélisée depuis de nombreuses années. Les méthodes d'apprentissage par renforcement s'entraînent d'abord en simulation le plus souvent avec une méthode dite "sans modèle" mais dans les faits, le modèle existe déjà dans le simulateur. Pourquoi ne pas l'exploiter directement dans la recherche de comportements ? Plusieurs travaux qui déploient des robots quadrupèdes dans des environnements accidentés (J. Lee et al., 2020 ; Miki et al., 2022) exploitent d'ailleurs le principe de démonstration en apprenant d'abord une politique privilégiée dans le simulateur puis en la distillant dans une politique robuste via un apprentissage par imitation.

Le fait de se restreindre à peu de biais d'apprentissage pose aussi problème au niveau de l'utilisateur. D'une part, les méthodes d'apprentissage par renforcement profond sont souvent restreintes à un apprentissage en simulation du fait de leur faible efficacité en échantillonnage. En effet, le monde réel ne peut pas être accéléré et la parallélisation de robots réels est très coûteuse. Il existe des méthodes qui essaient d'y pallier, par exemple pour apprendre à un quadrupède à marcher uniquement dans le monde réel en moins de 2 heures (S. Ha et al., 2020). D'autre part, chaque exécution de l'algorithme nécessite une nouvelle récolte de données, ce qui est souvent la partie la plus coûteuse en temps et en puissance de calcul aussi bien en simulation que dans le monde réel. Diviser l'apprentissage en une étape de collecte de données et une étape de résolution permet de minimiser le besoin de recollecter des données. De plus, la première phase de collecte peut profiter d'algorithme de résolution plus performant en exploration que l'apprentissage par renforcement, ce qui peut améliorer d'autant plus l'efficacité en échantillonnage.

Un point important à retenir de la robotique évolutionniste est l'utilisation de la diversité comportementale pour guider l'exploration (Mouret et Doncieux, 2009). MAP-Elites est un algorithme simple qui permet de générer un ensemble de comportements divers et de bonne qualité. Plusieurs approches utilisent MAP-Elites pour collecter un jeu de données qui est ensuite distillé dans une politique. Dans cette thèse, nous voulons concevoir une méthode similaire en séparant l'apprentissage en une phase de création d'un jeu de données puis une phase de généralisation de ce jeu de données pour obtenir une politique.

Chapitre 3

Trouver des solutions puis généraliser : apprendre un réflexe à un robot humanoïde endommagé

Contribution : Ce chapitre s'appuie sur le papier de journal *First Do Not Fall : Learning to Exploit a Wall With a Damaged Humanoid Robot* (Anne, Dalin et al., 2022). Ma contribution personnelle correspond à l'élaboration de l'algorithme d'apprentissage, de l'étude en simulation et du protocole de test sur le robot réel. Eloïse Dalin et Ivan Bergonzani étaient en charge du contrôleur du robot humanoïde et de déployer l'expérience sur le robot réel. Serena Ivaldi et Jean-Baptiste Mouret étaient les encadrants. Ce chapitre étend la publication avec une étude de la prédiction de la chute (Section 3.7.4).

Sommaire

3.1 Introduction

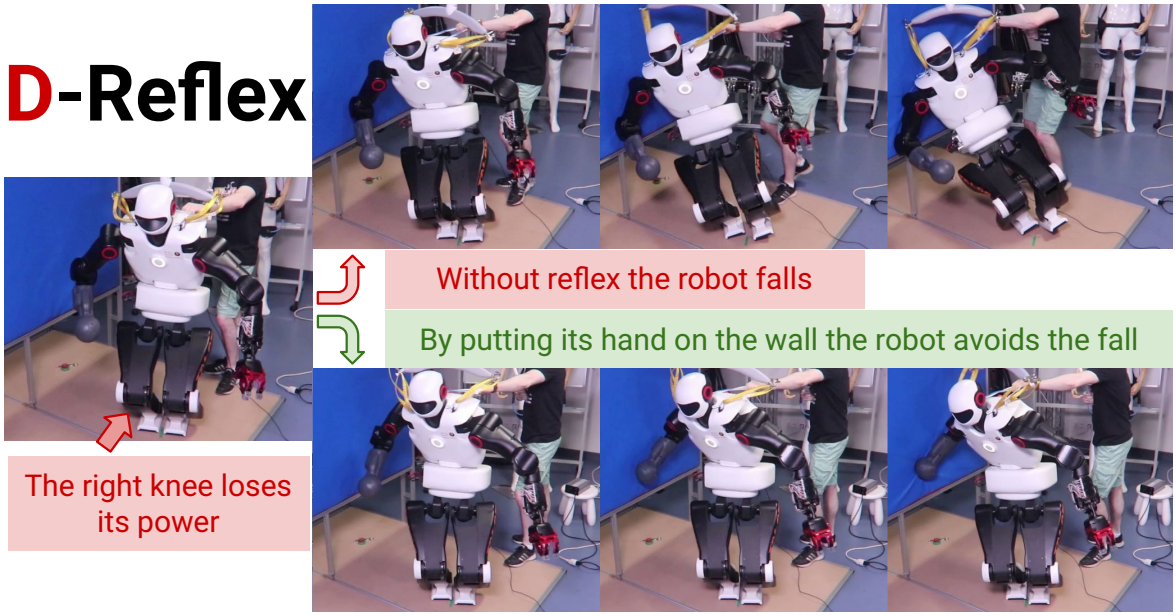

FIGURE 3.1 – Un robot humanoïde détecte une faute dans l'une de ses jambes. S'il ne réagit pas, il tombe ; mais il peut regagner sa stabilité en mettant sa "main" sur le mur à une position appropriée (qui dépend de sa posture et de la distance et l'orientation du mur).

Les robots humanoïdes sont des machines qui ont pour but de reproduire la polyvalence des êtres humains (Goswami et al., 2018). Elles sont ainsi les machines idéales à déployer dans des situations dangereuses et compliquées. Par exemple des désastres industriels ou des opérations spatiales, durant lesquelles plus de polyvalence signifie plus de chances d'avoir les capacités nécessaires pour mener à bien la mission (Goswami et al., 2018 ; Atkeson et al., 2018). C'est à mettre en opposition aux robots industriels qui sont créés dans le but de réaliser le même ensemble de tâches de façon répétée dans un environnement bien défini.

Cette polyvalence des robots humanoïdes engendre aussi une augmentation de leur fragilité : ils ont plus d'articulations que la plupart des robots, et une seule défaillance peut souvent engendrer la chute du robot (Atkeson et al., 2018) (la figure 3.1 montre un robot tomber lorsque que l'actionneur du genoux perd son alimentation). Cette fragilité est d'autant plus importante que les robots humanoïdes seraient le plus souvent utiles dans des situations trop dangereuses pour un humain et donc toutes aussi dangereuses pour le robot. Un robot déployé sur le terrain a donc de fortes chances d'être endommagé durant sa mission.

L'approche traditionnelle pour pallier les dommages d'un robot consiste à identifier le dommage, à mettre à jour le modèle du robot et ensuite utiliser ce modèle à jour pour réaliser la tâche (Hollerbach et al., 2008). Un robot humanoïde doit réagir rapidement pour prendre une décision avant que la chute ne devienne inévitable (Figure 3.2), tandis qu'identifier le modèle de la dynamique d'un robot aussi redondant requiert de nombreuses données et des trajectoires spécifiques (Hollerbach et al., 2008 ; Ramuzat et al., 2020). Un certain nombre de méthodes de rétablissement après dommages basées sur l'apprentissage pour des robots à deux pattes (Spitz et al., 2017), quatre pattes (Kaushik et al., 2020 ; Anne, Wilkinson et al., 2021) ou six pattes (Cully, Clune et al., 2015 ; Chatzilygeroudis et Mouret, 2018 ; Nagabandi, Clavera et al., 2019) ont été publiées récemment. Elles supposent toutes que le robot peut essayer un comportement, chuter, et ressayer jusqu'à ce qu'il trouve une façon de pallier les dommages. D'une part, les robots humanoïdes ne peuvent souvent pas se permettre de tomber sans risquer des dommages supplémentaires. D'autre part, une méthode d'essai-erreur devrait nécessiter la capacité de se remettre debout sans la connaissance du modèle du robot, ce qui est très difficile.

Dans ce chapitre, nous allons développer une méthode pour permettre à un robot humanoïde d'éviter la chute en s'appuyant contre un mur en cas de dommages (Figure 3.1). La méthode exploite le principe consistant d'abord à résoudre un ensemble de sous-problèmes pour ensuite généraliser les solutions trouvées et nous permet d'étudier la viabilité de ce principe d'apprentissage pour un robot réel. Une fois que le robot s'est stabilisé, un autre algorithme pourra être mis en place afin d'estimer l'état du robot endommagé et mettre à jour le modèle utilisé par le contrôleur. Ceci pourrait permettre, par exemple, d'imaginer un scénario où un robot humanoïde subit un dommage à une jambe et s'aide ensuite d'un mur pour continuer à avancer sur une jambe.

Notre première hypothèse de travail est qu'il est possible de détecter l'apparition d'une erreur dans la jambe sans pour autant savoir la source et la nature de l'erreur (erreur de capteur, dommage externe, perte d'une partie de la jambe, alimentation déconnectée, moteur bloqué). Ceci permet de rester général, en effet, une erreur quelconque à la jambe d'un robot humanoïde a de fortes chances de le faire tomber, et l'arrêt temporaire de la mission pour la recherche d'un point de support supplémentaire semble le plus pertinent. La seconde hypothèse de travail est que nous avons connaissance d'un mur à portée du robot et que nous connaissons connaît son orientation et son placement par rapport au robot. Cette hypothèse est raisonnable car la détection de plans à partir de capteurs visuels (par exemple, une caméra de capture de profondeur (Z. Zhang, 2012)) est aujourd'hui un problème résolu (Poppinga et al., 2008 ; M. Y. Yang et al., 2010).

À l'aide de ces deux hypothèses, notre méthode D-Reflex (pour *Damage-Reflex*) permet de trouver en quelques millisecondes un point de contact sur le mur et permet, grâce au contrôleur corps complet, de retrouver un état stable.

L'idée principale de D-Reflex est d'entraîner un réseau de neurones à estimer les chances de succès de chaque point de contact sur le mur étant donné la posture du robot et la configuration du mur par rapport au robot. Pour entraîner ce réseau de neurones, nous réalisons un apprentissage en deux étapes : une première étape de résolution d'un grand nombre de situations en simulation et une seconde étape d'apprentissage supervisé des positions de contact trouvées pour ces situations.

Séparer le processus en deux étapes — collecte des données et entraînement du réseau de neurones — permet d'exploiter au mieux les techniques matures d'apprentissage supervisé tout en séparant la simulation et l'apprentissage. Une fois les données collectées, il est tout à fait possible de tester différentes techniques d'apprentissage sans avoir à relancer une seule

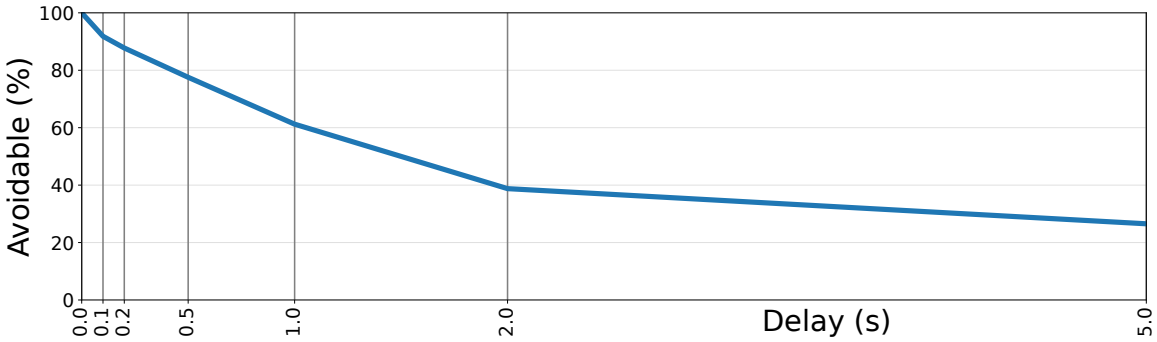

FIGURE 3.2 – Le pourcentage de chutes évitées (Section 3.5.2 pour la définition "d'évitable") décroît quand le délai entre le déclenchement du dommage et le déclenchement du réflexe augmente ; le robot n'a que quelques millisecondes pour réagir avec un haut taux de succès.

simulation. Le réseau de neurones, une fois entraîné, peut être interrogé en quelques millisecondes et en parallèle, ce qui est idéal pour un réflexe d'urgence dans le cas d'un robot en chute.

Nous présentons des expériences effectuées en simulation et sur un réel robot humanoïde TALOS (1,75 m, 100 kg, 30 degrés de liberté).

3.2 Formulation du problème

Un robot humanoïde subit un dommage à l'une de ses jambes, ce qui a de grandes chances de le faire tomber. Le but est de trouver une position de contact pour la main se trouvant du côté de la jambe endommagée afin d'éviter la chute et de retrouver un état stable (Figure 3.1).

Les informations à la disposition de l'algorithme sont :

— une des jambes à subi un dommage qu'il est possible de détecter en supposant que l'articulation endommagée renvoie une erreur ou ne répond plus ;
— il y a un mur (surface plane verticale) à portée de bras du robot dont la distance d et l'orientation α par rapport au robot sont connues (cette position peut facilement être connue à l'aide d'une caméra de capture de profondeur (Z. Zhang, 2012) montée sur l'épaule du robot (Poppinga et al., 2008 ; M. Y. Yang et al., 2010)) ;
— la posture q du robot au moment de la détection du dommage.

Les informations inconnues sont :

— quelles sont les articulations endommagées J et la nature des dommages ;
— les positions de contacts sur le mur qui permettraient de retrouver l'équilibre.

La fonction recherchée est la fonction $f : (q, d, \alpha) \rightarrow (x^*, y^*)$ qui renvoie une position de contact robuste à la nature de dommage. Nous faisons l'hypothèse que le robot à un contrôleur qui permet à la main d'atteindre cette position.

3.2.1 Le robot humanoïde TALOS

Le robot humanoïde utilisé pour l'ensemble des expériences est le robot TALOS (Stasse et al., 2017). Il s'agit d'un robot humanoïde de 1,75 m pour 100 kg et doté de 32 degrés de liberté : 7 pour chaque bras, 6 pour chaque jambe, 2 pour la tête, 2 pour le torse et 1 pour chaque pince (Figure 3.3).

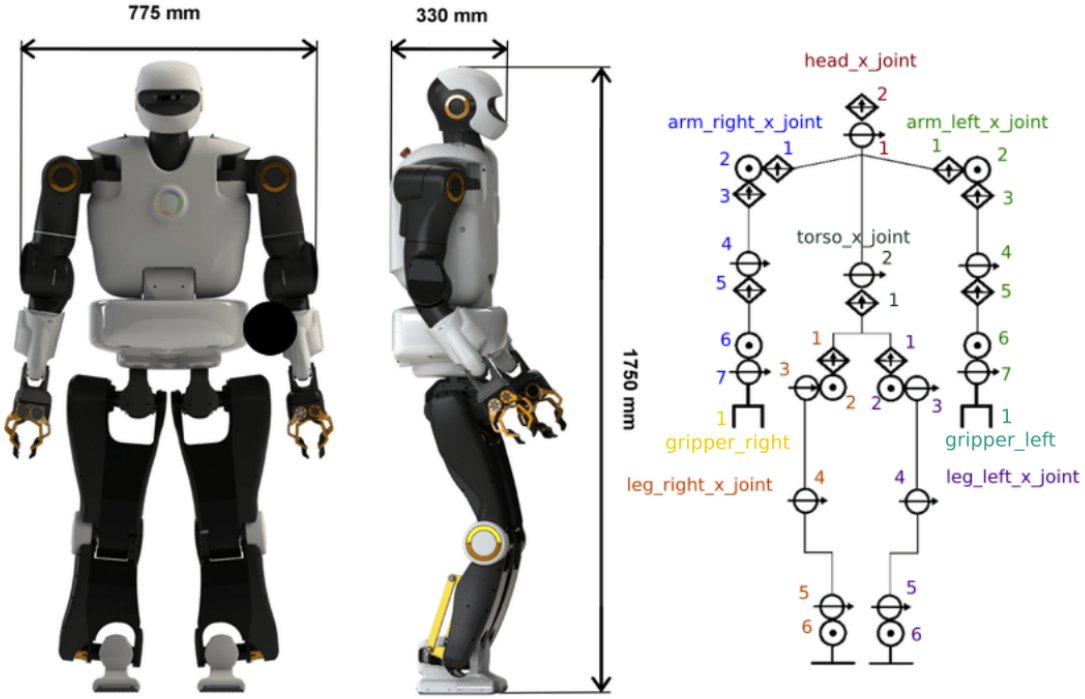

FIGURE 3.3 – Vue schématique des degrés de liberté du robot humanoïde TALOS (Stasse et al., 2017).

3.2.2 Les dommages considérés dans cette étude.

Un robot humanoïde peut être endommagé de multiples façons et il est souvent difficile de déterminer précisément la nature du dommage. Pour cette étude, nous utilisons trois types de dommages : l'amputation (une partie de la jambe est manquante), des actionneurs passifs (l'actionneur ne délivre aucun couple et l'articulation peut tourner librement dans son axe de rotation) et un actionneur bloqué (la position de l'articulation est fixée). Les deux premiers dommages sont dangereux pour l'équilibre du robot qui utilise le plus souvent ses deux jambes pour rester en équilibre. Ils résultent le plus souvent en une chute du robot mais ne sont pas facilement distinguables du point de vue du robot ; dans les deux cas, la carte électronique ne répond plus ou renvoie une erreur.

3.2.3 Représentation du mur

Pour ce chapitre, nous étudions l'utilisation d'un réflexe utilisant la main droite pour chercher appui sur un mur. Par mur, nous entendons une surface plane et verticale. Nous faisons l'hypothèse qu'étant donné la portée du robot, le mur est de hauteur et longueur infinies, ce qui permet de décrire sa configuration avec uniquement deux paramètres. En effet, tout solide rigide est défini par sa position et son orientation, soit 6 degrés de liberté. En supposant le mur infini sur sa hauteur (l'axe des Z) et sa longueur (l'axe des X), il est donc invariant en translation selon ces axes et en rotation selon l'axe des Y. De plus, comme le mur est supposé vertical, cela fixe deux degrés

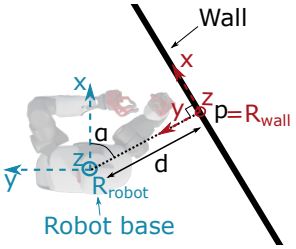

FIGURE 3.4 – Vue schématique du mur.

de liberté de rotation, le mur ne pouvant tourner que selon l'axe
des Z. Ce qui ne laisse donc que deux degrés de liberté : la rotation autour de l'axe Z et la translation selon l'axe Y.

Si nous définissons le point du mur p le plus proche de la base flottante R du robot. Alors la configuration du mur est entièrement caractérisée par la distance $d = ||p - R||_2$ et l'angle α entre l'axe des X (dans la base du robot) et le vecteur $\vec{p} = \vec{Rp}$ (Figure 3.4).

3.2.4 Étude du délai pour réagir

Nous avons réalisé une étude préliminaire pour estimer le temps qu'a le robot entre le déclenchement du dommage et le moment où la chute (et les dommages supplémentaires qu'elle engendre) deviennent inévitables. Pour ce faire, nous avons utilisé le même système de collecte de données pour trouver des solutions, c'est-à-dire des positions de contacts qui permettent au robot de regagner son équilibre en s'appuyant sur le mur, mais en déclenchant le réflexe après un certain temps (entre 0 s et 5 s).

La figure 3.2 présente le pourcentage de chutes évitées en fonction de ce délai pour un ensemble de 50 situations pour lesquelles nous savons qu'il existe au moins une solution avec un délai nul. Le robot n'a que quelques millisecondes (c'est-à-dire moins de 100 millisecondes) pour réagir avec un haut taux de succès (c'est-à-dire supérieur à 90%).

3.3 État de l'art

3.3.1 Robotique

3.3.1.1 Contrôle Corps Complet

Le contrôle du corps complet est une méthode basée sur l'optimisation quadratique pour contrôler un robot humanoïde de façon à exploiter la forte redondance de ses degrés de liberté (Bouyarmane et al., 2017 ; Dalin et al., 2021).

À chaque pas de temps, le contrôleur optimise les accélérations $\ddot{q}$ et les couples τ de l'ensemble des articulations pour minimiser une somme de fonctions de coût quadratiques contrainte par la dynamique du modèle (une contrainte d'égalité) et d'autres contraintes d'égalité et d'inégalité :

$$\begin{aligned}(\tau^*, \ddot{q}^*) = \underset{\tau,\ddot{q}}{argmin} & \sum_{k=0}^{n\_tasks} w_k \|A_k(q,\dot{q})\ddot{q} - b_k(q,\dot{q})\|^2 \\ \text{tel que } & A_{ineq}(q,\dot{q})\ddot{q} \leq b_{ineq}(q,\dot{q}) \\ \text{tel que } & A_{eq}(q,\dot{q})\ddot{q} = b_{eq}(q,\dot{q}) \\ \text{tel que } & \tau = M(q)\ddot{q} + F(q,\dot{q})\end{aligned} \tag{3.1}$$

où w_k est le poids de la tâche k décrite par $A_k(q,\dot{q})$ et $b_k(q,\dot{q})$, M est la matrice de masse dans l'espace des articulations et F contient tous les termes non linéaires (la force de Coriolis, la force centrifuge, la gravité et les contacts).

Dans ce travail, nous utilisons le contrôleur décrit dans Dalin et al. (2021). Les tâches principales sont :

- les tâches de position cartésienne et orientation des mains et des pieds ;
- la tâche de position du centre de masse ;
- la tâche posturale par défaut qui permet d'assurer l'unicité du problème d'optimisation quadratique et de biaiser les solutions vers une position "neutre".

Les contraintes sont : les contact entre les pieds et le sol, et le respect des limites articulaires en position, vitesse et accélération.

Une fois que le couple optimal et les accélérations des articulations ont été calculés, elles sont intégrées en utilisant le modèle du robot afin d'obtenir les positions articulaires désirées, qui sont passées au contrôleur de bas niveau de chaque articulation.

Pour le robot réel TALOS, les contrôleurs de bas niveau sont des contrôleurs PID (Stasse et al., 2017). En simulation, nous utilisons un contrôleur Stable-PD (Tan, K. Liu et al., 2011) qui prend en compte le modèle du robot pour calculer les couples envoyés en fonction de la position articulaire désirée. Ce contrôleur fournit, selon notre expérience,une bonne approximation d'un contrôleur PID bien paramétré.

3.3.1.2 Planification Multi-Contact

La planification multi-contact (par exemple, savoir où mettre la main sur le mur) fait partie des sujets de recherche en robotique humanoïde (Kheddar et al., 2019 ; Polverini et al., 2021). Cependant, les algorithmes de planification tendent à nécessiter plus que les quelques instants qu'un robot humanoïde peut se permettre (par exemple, environ 100 ms pour planifier un mouvement sur un contact cible (Mastalli et al., 2020)). À cela s'ajoutent deux problèmes. D'une part, il faut savoir où situer le contact cible. C'est toujours un problème ouvert, qui peut par exemple être résolu par un expert comme en téléopération (Rouxel et al., 2023). D'autre part, la plupart des algorithmes de planification font l'hypothèse d'un modèle connu de la dynamique du robot, ce qui n'est pas le cas pour un robot endommagé et d'autant plus lors d'une dynamique de chute.

3.3.2 Adaptation aux dommages et aux chutes

3.3.2.1 Identification des Dommages

Une étape préliminaire à tout algorithme de planification est donc l'identification du modèle du robot endommagé (Hollerbach et al., 2008) qui correspond à estimer ses paramètres cinématiques et inertiels. Identifier un modèle de la dynamique d'un robot humanoïde requiert au moins plusieurs minutes de collecte de données et des trajectoires "excitantes" spécifiques. Par exemple, l'identification de l'articulation du coude du robot TALOS nécessite 230 s de collecte de données sur une trajectoire "excitante" bien choisie (Ramuzat et al., 2020). Il semble bien optimiste de pouvoir identifier un nouveau modèle en quelques millisecondes de données des capteurs internes au robot.

3.3.2.2 Étude de la chute

En s'inspirant de la façon dont les humains réagissent quand ils tombent face à un mur, Cui et al. (2021) ont récemment proposé de placer les bras du robot de façon à maximiser l'ellipsoïde de raideur, augmentant ainsi la stabilité et la capacité d'absorption du choc. En comparaison de nos travaux, ils font l'hypothèse d'un mur uniquement frontal et, plus important, que le modèle du robot est connu et non endommagé.

3.3.2.3 Mitigation des dommages

Quand la chute est inévitable, plusieurs méthodes ont été démontrées comme capables de mitiger les dommages sur différents humanoïdes, en distinguant plusieurs phases de la chute :

— durant la phase "pré-impact", le robot adapte sa posture pour éviter des postures de "singularité de chute" qui augmentent l'impact (Samy et Kheddar, 2015), en cherchant une posture sûre en tombant sur le dos (Q. Li et al., 2017), ou en réalisant une stratégie de roulade (D. Liu et al., 2021) ;
— durant la phase "d'impact", les gains des contrôleurs PID sont automatiquement adaptés en les incluant à l'intérieur du contrôleur QP pour empêcher les actionneurs d'atteindre leurs limites en couple tout en atteignant la posture désirée (Samy, Bouyarmane et al., 2017) ;
— Durant la phase "post-impact", une optimisation quadratique des forces distribue la quantité de mouvement excédentaire accumulée pendant la chute dans les différentes parties du corps (Samy, Caron et al., 2017).

Ces méthodes ne cherchent pas à éviter la chute mais à mitiger les dommages induits par la chute. Globalement, ces travaux sont complémentaires aux nôtres : bien que de nombreuses chutes puissent être évitées en ajoutant un point de contact avec un mur, certaines chutes ne peuvent pas être évitées et dans ce cas le seul comportement restant consiste à minimiser les dommages.

3.3.2.4 Apprentissage pour pallier des dommages

Une approche prometteuse pour permettre aux robots de s'adapter à des dommages imprévus consiste à apprendre le modèle de la dynamique en utilisant des méthodes d'apprentissage générique, comme un réseau de neurones ou un processus gaussien. Pour minimiser la quantité de données nécessaire, les méthodes actuelles utilisent une simulation du robot intact (Chatzilygeroudis et Mouret, 2018) ou du méta-apprentissage pour entraîner un modèle rapidement (Nagabandi, Clavera et al., 2019 ; Kaushik et al., 2020 ; Anne, Wilkinson et al., 2021). Des expériences réussies sur des robots à quatre pattes (Kaushik et al., 2020 ; Anne, Wilkinson et al., 2021) et six pattes (Chatzilygeroudis et Mouret, 2018 ; Nagabandi, Clavera et al., 2019) ont été publiées, mais, à notre connaissance, aucune avec un robot bipède. Une importante différence entre les robots humanoïdes et les autres robots à pattes est que ces derniers peuvent se permettre de tomber pendant l'apprentissage car se remettre sur leurs pattes leur est plus facile.

Spitz et al. (2017) ont regardé comment un robot humanoïde endommagé peut adapter son comportement pour éviter des états déjà rencontrés qui ont mené à une chute. De façon similaire, Cully, Clune et al. (2015) ont développé un algorithme d'apprentissage pour qu'un robot à six pattes endommagé trouve un nouveau comportement qui ne s'appuie pas sur les parties endommagées. Dans ces deux cas, les auteurs utilisent un apprentissage épisodique dans lequel le robot peut essayer un comportement, chuter, et réessayer encore plusieurs fois. Ce type d'apprentissage épisodique n'est pas applicable quand l'objectif est d'éviter la chute immédiatement après avoir subi des dommages.

À notre connaissance, le travail le plus proche du nôtre consiste en un robot quadrupède qui effectue des acrobaties "comme un chat" pour atterrir sur ses pattes pendant une chute de plusieurs mètres (Kurtz et al., 2021). Pour résoudre ce problème, un algorithme d'optimisation génère d'abord un jeu de données d'exemples en simulation puis entraîne un réseau de neurones à imiter ces réflexes de façon à être suffisamment rapide pour pouvoir être déployé sur le robot. De façon similaire, nous cherchons à résoudre un problème de façon exhaustive en simulation puis utilisons ces résultats pour entraîner un réseau de neurones rapide qui permet de choisir le comportement approprié.

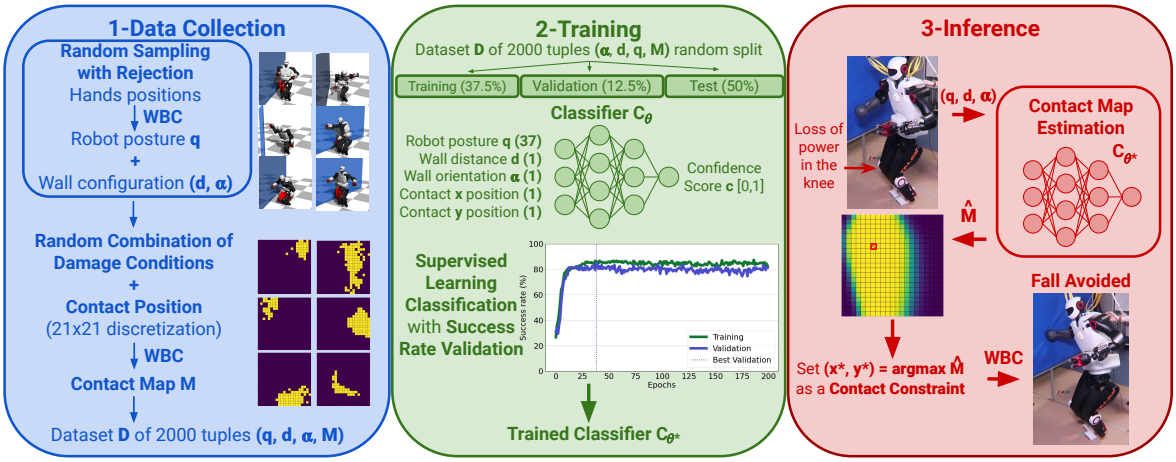

FIGURE 3.5 – Vue globale de D-Reflex. La première étape (Collecte de données) consiste à échantillonner différentes situations : configurations du mur (distance et orientation), postures du robot humanoïde et combinaisons de dommages, et à simuler le comportement du robot pour chaque position de contact sur une grille de 21×21 sur le mur (soit 441 simulations par situation). La seconde étape (Apprentissage) consiste à entraîner un réseau de neurones de classification pour prédire le succès (éviter la chute et retrouver l'équilibre) pour chaque position de contact de la grille. La troisième étape (Inférence) consiste à interroger ce réseau de neurones entraîné pour chaque point de la grille dans une nouvelle situation afin de sélectionner quelle position de contact utiliser pour le réflexe.

3.4 Méthode

3.4.1 Collecte de données

Cette première partie consiste à générer un grand nombre de situations différentes et à évaluer pour chacune un grand nombre de positions de contact sur le mur (Figure 3.5.1).

Une situation correspond à : une posture du robot q, une configuration du mur (distance d et orientation α) ainsi qu'à une combinaison de dommages.

3.4.1.1 Échantillonnage Aléatoire des postures

Pour échantillonner différentes postures atteignables q du robot, l'algorithme échantillonne d'abord une position cartésienne dans un cube autour de chaque main et exécute le contrôleur corps complet pendant 4 s pour positionner les deux mains à ces positions. Ceci permet d'avoir des postures différentes (par exemple, baissées, sur le côté ou de face) et réalisables par le robot de façon plus rapide que d'échantillonner directement dans l'espace des articulations.

3.4.1.2 Échantillonnage de la position du mur

L'objectif est d'échantillonner des configurations du mur (une distance d et une orientation α) qui sont à portée du robot et qui n'engendrent pas de collision avec le mur avant le déclenchement du dommage. Pour ce faire, l'algorithme exécute les 4 premières secondes du mouvement pendant lesquelles le robot déplace juste ses mains et rejette les configurations qui engendrent un contact entre le robot et le mur.

3.4.1.3 Échantillonnage de la combinaison de dommages

La jambe du robot étant composée de 6 articulations, l'algorithme choisit pour chacune la présence ou non d'un dommage et sa nature : amputation, passive ou bloquée (sachant que si le robot subit une amputation, les articulations enfants sont automatiquement amputées). Afin de mieux répartir les différents types de dommages, un poids différent est donné pour chaque type. La figure 3.6 présente la distribution des différentes combinaisons utilisées dans cette thèse.

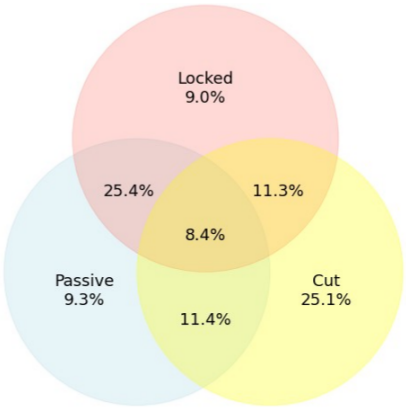

FIGURE 3.6 – Distribution des différentes combinaisons de dommages.

3.4.1.4 Construction de la carte des contacts

L'algorithme discrétise le mur en une grille bidimensionnelle et évalue en simulation chacune de ses positions de contact. Le robot est donc d'abord placé à la posture de la situation en exécutant le contrôleur corps complet pendant 4 s, puis le dommage est déclenché et la position de contact sur le mur est envoyée au contrôleur corps complet sous forme d'une tâche cartésienne de contact entre la main et une surface plane. Le contrôleur corps complet optimise l'ensemble de la posture du robot pour essayer de satisfaire cette tâche supplémentaire.

L'algorithme attend ensuite 11 s (ce qui donne un épisode complet de 15 s) avant de valider que le robot a retrouvé son équilibre. Dans le cas où le robot tombe, c'est-à-dire si le robot est entré en contact avec le mur avec autre chose que ses mains, ou avec le sol avec autre chose que ses pieds, entraînant un arrêt de la simulation, alors la position de contact est un échec. Dans le cas contraire, c'est-à-dire si le robot est toujours debout au bout de 11 s (soit 15 s au total), la position de contact est un succès.

Nous ne considérons que des contacts avec la main afin d'avoir des données plus propres et de diminuer les artefacts de contact dus à la simulation. De plus, le robot TALOS est doté de capteurs de force et de couple au niveau du poignet qui peuvent être utilisés pour détecter le contact au moment du réflexe et qui pourront potentiellement servir pour identifier l'état du robot lorsque celui-ci est redevenu stable.

L'algorithme collecte donc pour chaque situation une matrice booléenne que nous appellerons carte de contacts $M \in 0, 1^{(n,n)}$ où n correspond à la précision de la discrétisation. La figure 3.5.1 montre différents exemples de carte de contacts.

3.4.2 Apprentissage

Une fois que l'algorithme a collecté un jeu de données D d'échantillons (q, d, α, M), il le sépare en un jeu d'entraînement (37,5%), un jeu de validation (12,5%) et un jeu d'évaluation (50%).

L'objectif est d'apprendre une fonction

$$f : (q, d, \alpha) \rightarrow (x^*, y^*)$$

où (x^*, y^*) est une position de contact réussie dans le référentiel du mur. Comme la carte de contacts M contient souvent plus d'une réussite, elle ne permet pas de directement obtenir f.

À la place, l'algorithme apprend une fonction de classification qui prédit le succès de chaque position de contact :

$$C : (q, d, \alpha, x, y) \rightarrow c$$

où $c \in [0, 1]$ représente la confiance que la position de contact (x, y) soit un succès.

Pour estimer la fonction de classification C, l'algorithme entraîne un réseau de neurones $\hat{C}_\theta$ paramétré par un ensemble de poids θ en utilisant PyTorch (Paszke et al., 2019) avec comme fonction d'erreur l'entropie croisée et l'optimiseur Adam (Kingma et al., 2014).

Une étude par recherche aléatoire des hyperparamètres a permis de fixer l'architecture du réseau de neurones avec deux couches cachées entièrement connectées de 1024 neurones avec la fonction d'activation *Rectified Linear Unit* (ReLU) (Agarap, 2018), un taux de suppression de neurones (*dropout*) de 0,2 et un taux d'apprentissage de 10^{-5}.

Pour obtenir une estimation de la fonction f, l'algorithme interroge le réseau de neurones aux positions $(x, y) \in (X, Y)$ sur le mur et sélectionne la position avec la meilleure prédiction de succès.

$$\hat{f}(q, d, \alpha) = \underset{(x,y)\in(X,Y)}{argmax}\ \hat{C}_\theta(q, d, \alpha, x, y).$$

Comme les positions sont bidimensionnelles et qu'il n'y a pas besoin d'une précision millimétrique, l'algorithme réalise cette optimisation à l'aide d'une recherche par grille. Plus précisément, l'algorithme utilise une grille de même résolution 21×21 ce qui correspond à 441 appels du réseau de neurones. Cela correspond à peu près à 4 ms sur une Nvidia RTX 2080 (utilisée pour les évaluations en simulation) et à environ 16,7 ms [min: 14, 0 ms, max : 21, 9 ms] sur l'un des CPUs inclus dans le Talos.

Un entraînement classique supervisé évalue de façon périodique le score de prédiction du réseau de neurones sur le jeu de validation afin d'éviter le sur-apprentissage. Ici, l'utilité du classifieur se mesure par son efficacité à sélectionner une position de contact réussie. En d'autres termes, l'algorithme évalue le réseau de neurones en regardant $\hat{f}(q, d, \alpha)$ sur le jeu de validation plutôt que $\hat{C}_\theta(q, d, \alpha, x, y)$ car le plus important est d'évaluer l'efficacité finale de la méthode et non pas le réseau de neurones lui-même.

À la fin de l'apprentissage, l'algorithme sélectionne l'ensemble de poids θ correspondant au plus haut taux de succès sur le jeu de validation (Figure 3.5.2).

3.4.3 Inférence

Le scénario d'utilisation du réflexe est le suivant : lors de sa mission, le robot réalise sa mission en (1) interrogeant périodiquement les cartes de contrôle de ses articulations pour détecter le moindre problème et (2) en détectant les murs proches dans son environnement de façon à toujours connaître le plus proche. Pour cette première étude, nous supposons que ces données sont fournies par un capteur dédié.

Lorsqu'il détecte une erreur à sa jambe, il utilise sa dernière posture connue q, la distance d et l'orientation α du mur le plus proche et interroge le modèle entraîné $\hat{C}_\theta$ sur une grille de positions pour estimer $(x^*, y^*) = \hat{f}(q, d, \alpha)$. Figure 3.7 montre des exemples de prédiction de carte de contacts.

Puis, il ajoute cette position comme tâche de contact dans le contrôleur corps complet afin qu'il amène le robot à mettre sa main sur la position choisie (Figure 3.5.3).

3.4.4 Détails d'implémentions

3.4.4.1 Déclenchement du réflexe

Nous supposons que le robot est capable de détecter lorsque l'un de ses actionneurs a une faute afin de déclencher immédiatement le réflexe. Dans ce cas, le robot se place dans un mode "survie" qui enlève toutes les tâches non essentielles liées à la mission en cours. Il ne garde que les contraintes de contact des pieds, les contraintes des limites articulaires, la tâche de position cartésienne du centre de masse et la tâche posturale. Il ajoute ensuite une tâche de contact de la main à la position sélectionnée avec un poids élevé afin que le contrôleur corps complet génère la trajectoire pour l'atteindre.

La tâche du centre de masse requiert le choix d'une position cartésienne. Par défaut, celle-ci correspond à un point entre les deux pieds. Dans le cas étudié, comme nous prenons pour hypothèse qu'une des deux jambes est endommagée et qu'aucun poids ne doit y être attribué, le centre de masse doit se déplacer entre la jambe non endommagée et le point de contact sur le mur. Comme le robot ne peut connaître avec précision la position du contact (le capteur de force et de couple ne se situe qu'au poignet) et qu'il ne connaît pas lors du réflexe son modèle endommagé, nous avons décidé après des expériences préliminaires de ne pas changer la cible de la tâche du centre de masse car elle a montré les meilleurs résultats. Lorsque le centre de masse est placé trop près du mur ou du pied restant, le taux de succès diminue rapidement.

Une meilleure solution serait d'inclure la cible du centre de masse comme variable d'optimisation.

3.4.4.2 Post-contact

À cause de la différence entre le robot endommagé et le modèle utilisé qui correspond à celui du robot non endommagé, la position de contact atteinte est rarement celle sélectionnée comme tâche dans le contrôleur corps complet (la section 3.7.5 apporte plus de détails). Pour prendre cela en compte, une fois que le contact est détecté, l'algorithme met à jour la tâche de contrainte en modifiant la position cible par la position courante de la main, ce qui permet de stopper les mouvements du robot. En effet, à cause de la chute, le robot endommagé se trouve plus proche du mur que le modèle utilisé par le contrôleur corps complet (qui lui ne chute pas), ce qui entraîne une élongation cible du bras plus importante que nécessaire, impliquant un appui plus fort sur le mur (à cause d'un contrôle en position bas niveau qui cherche à atteindre l'élongation cible) impliquant le plus souvent un rebond qui fait tomber le robot du côté opposé.

3.5 Expérimentation

L'expérience est réalisée sur le robot TALOS avec comme simulateur physique RobotDART [10]. Afin de ne pas abîmer les pinces du robot réel, nous avons monté une boule imprimée en 3D à la place de la main droite. Pour faciliter le passage de la simulation à la réalité, nous avons aussi remplacé cette main par une boule en simulation. Une étude plus approfondie pourrait par la suite étudier l'utilisation du poignet, du bras, du coude ou de l'épaule pour arrêter la chute avec moins de risque de dommages supplémentaires et en offrant possiblement un meilleur support.

10. https://github.com/resibots/robot_dart

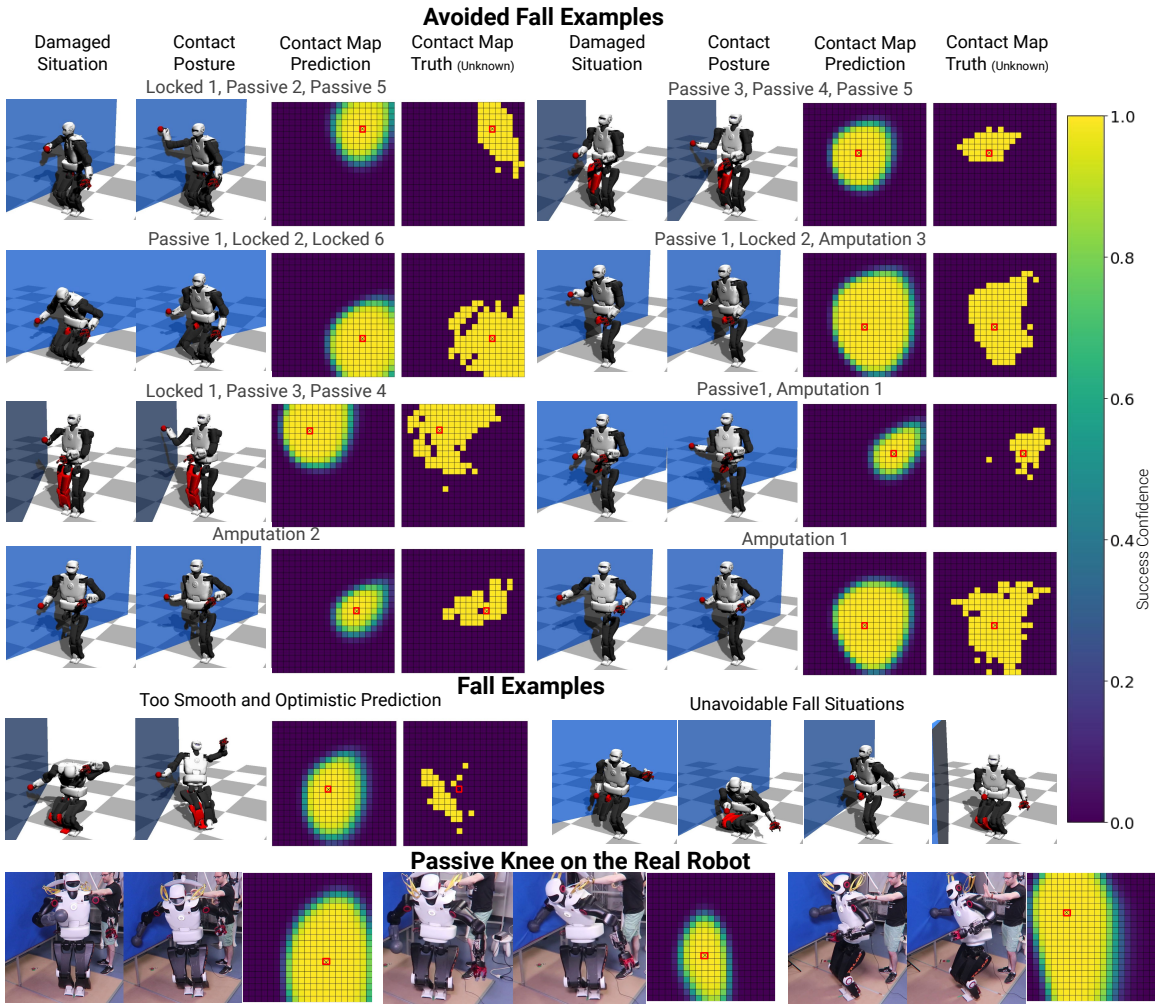

FIGURE 3.7 – Exemples de situations où le robot est endommagé, accompagnées des réflexes (réussis ou échoués) en utilisant D-Reflex sur le robot TALOS (Stasse et al., 2017) en simulation et sur le robot réel, la vidéo https://youtu.be/hbuWr-ZNAtg montre aussi différents exemples. La carte de contacts estimée par le réseau de neurones et la carte de contacts réelle (calculée en simulation mais inconnue au moment de l'inférence). Nous distinguons deux cas d'échecs : quand la prédiction de la carte de contacts est trop lisse et optimiste par rapport à la réalité, et quand la chute est tout simplement inévitable (par exemple, quand le robot est penché en avant ou que le mur est trop loin).

Les expériences sont réalisées uniquement sur le côté droit du robot, c'est-à-dire en endommageant uniquement la jambe droite et en plaçant le mur du côté droit du robot. L'hypothèse étant que, grâce à la symétrie du robot, le réseau de neurones entraîné peut être utilisé pour le côté gauche, c'est-à-dire en endommageant la jambe gauche et en plaçant le mur sur le côté gauche du robot. Afin de vérifier cette hypothèse, nous avons entraîné un D-Reflex sur le côté droit que nous avons évalué sur le côté gauche (en symétrisant les données d'entrée du réseau) et avons obtenu une correspondance de 98,5% sur 278 situations.

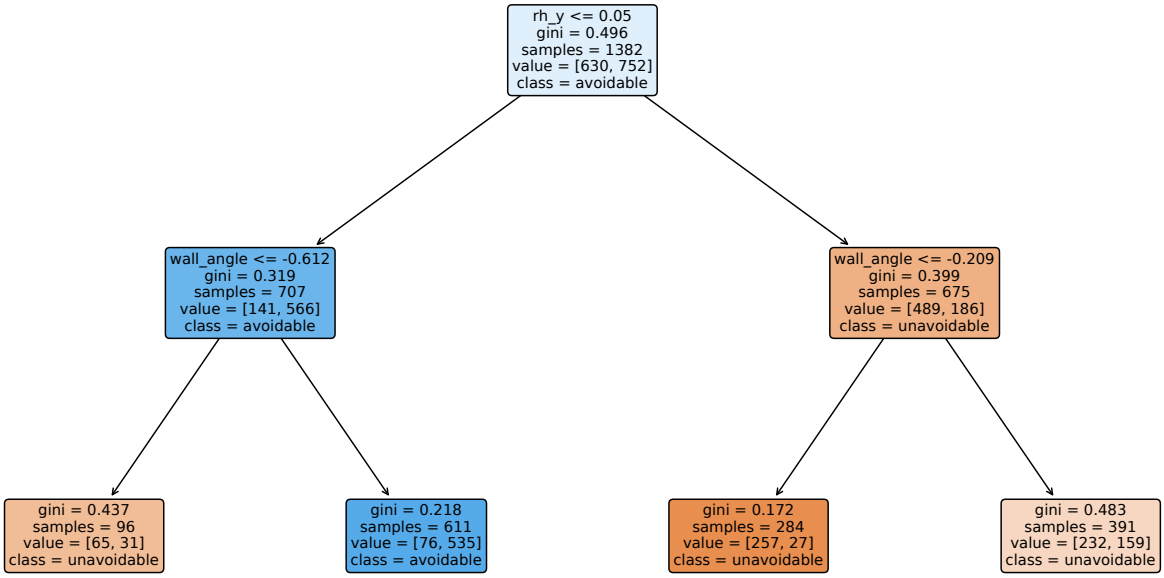

FIGURE 3.8 – Un arbre de décision pour classifier les situations ayant des chutes évitables et des chutes inévitables (en utilisant 1382 situations provenant du jeu de données de 2 000 situations collectées). Utiliser le décalage de la main sur l'axe des Y et l'orientation α du mur permet d'obtenir 78,8% d'exactitude de prédiction.

3.5.1 Valeurs des Paramètres

La posture initiale du robot est obtenue en sélectionnant des positions aléatoires des mains et en laissant le contrôleur corps complet placer le robot à ces positions pendant 4 s. La position aléatoire des mains est échantillonnée uniformément dans un pavé droit de taille : $[-0,1$ m, $+0,2$ m] dans l'axe sagittal, $[-0,4$ m, $+0,4$ m] dans l'axe frontal et $[-0,5$ m, $+0,4$ m] dans l'axe longitudinal.

La distance du mur est entre $0,4$ m et 1 m (le bras faisant environ 80 cm depuis l'épaule). L'orientation est échantillonnée dans $[-1,1]$ rad.

La carte de contacts créée utilise 21 positions dans les deux axes horizontal et vertical, ce qui résulte en 441 positions différentes pour chaque situation. Les dimensions sont de $-0,75$ m à $+0,75$ m sur l'axe horizontal et de $-0,5$ m à $+0,75$ m sur l'axe vertical. La figure 3.7 montre des exemples de carte de contacts.

Le jeu de données est constitué de 2 000 situations différentes (posture du robot, configuration du mur et combinaison de dommages de la jambe droite) et de 441 simulations de contact pour chacune (4 s de mise en place du robot, déclenchement du réflexe puis attente d'équilibre pendant 11 s), ce qui correspond donc à 882 000 simulations de 15 s (ce qui a nécessité un peu moins de 3 jours de calcul en parallèle sur un ordinateur multicœur).

Le code source et le jeu de données de 882 000 simulations sont disponibles en ligne [11].

3.5.2 Évaluation

Une analyse des cartes de contacts montre que 31,5% des situations du jeu de donnée ne contiennent pas de position de contact qui permettent de se prémunir de la chute. La figure 3.7 montre 4 exemples de situations dont la chute n'est pas évitable.

11. https://github.com/resibots/d-reflex

En apprenant un arbre de décision pour classifier les situations où la chute n'est pas évitable de celles où la chute est évitable (Figure 3.8), nous avons trouvé deux critères discriminants : le décalage de la main par rapport à la position initiale sur l'axe des Y et l'angle du mur. En effet, si la main est trop loin du mur (c'est-à-dire le décalage de la main sur l'axe des Y est supérieur à $0,05$ cm) ou si le mur est orienté vers l'arrière du robot (c'est-à-dire $\alpha \leq -0,612$ rad), alors la chute ne peut être évitée en mettant la main sur le mur.

De plus, pour une partie importante des cartes de contacts (618 situations sur les 2 000, soit 30,9%), il n'y a seulement qu'un petit nombre de positions de contact sur la carte de contacts qui sont réussies. Nous préférons considérer que ces positions de contacts ont peu de chance d'être réalisables en pratique et sont le plus souvent dues à une combinaison "d'événements chanceux" extrêmement sensibles aux détails de la simulation. En enlevant ces cartes de contact peu réalistes, il reste 37,6% des situations (c'est-à-dire 752 sur les 2 000) pour lesquelles nous avons des solutions réalistes. En conséquence, nous définissons une chute comme étant "évitable" si et seulement si la carte de contacts de la situation correspondante possède au moins 9 positions de contact voisines (sous forme d'une sous-grille 3×3) qui sont réussies.

Pour l'évaluation des différentes méthodes, nous nous intéressons uniquement à ces situations dites évitables étant donné qu'il est facile de générer une proportion arbitraire de situations non évitables.

Pour assurer que nos résultats ne dépendent pas d'une seule évaluation chanceuse, nous avons répliqué la phase d'entraînement 20 fois sur 20 différentes séparations du jeu de données (entraînement, validation et évaluation) avec différentes graines aléatoires.

3.5.3 Méthodes de référence

Nous comparons notre méthode, **D-Reflex**, à deux méthodes de référence :

— **Sans Réflexe** : le robot ne réagit pas ("Est-il nécessaire de déployer un réflexe ?") ;

— **Réflexe aléatoire** : le robot sélectionne une position de contact uniformément parmi les 441 possibilités ("Est-ce que le problème est trivial ? Est-ce qu'il y a un besoin d'apprendre ?").

3.5.4 Ablations et Additions

Pour mieux comprendre la nécessité de chaque élément de la méthode, nous nous comparons aussi à 3 ablations du réseau de neurones, qui correspondent à apprendre un réseau de neurones avec un sous-ensemble des données d'entrée :

— **Ablation de la Posture** : la posture q est enlevée de l'entrée du réseau de neurones. ("Est-ce que la position de contact dépend de la posture du robot ?") ;

— **Ablation du Mur** : la configuration du mur (la distance d et l'orientation α) est enlevée de l'entrée du réseau de neurones. ("Est-ce que la position de contact dépend de la configuration du mur ?") ;

— **Ablation Totale** : la position de contact est indépendante de la posture du robot et de la configuration du mur, ce qui correspond à simplement sélectionner dans le jeu d'entraînement la meilleure position de contact en moyenne, correspondant plus ou moins au centre du mur ("Est-ce qu'il existe une solution qui fonctionne pour toutes les situations ?").

Pour vérifier que nous n'avons pas ignoré des informations pertinentes, nous nous comparons aussi à deux additions, ce qui correspond à apprendre un autre réseau de neurones avec des données supplémentaires :

— **Addition** J : l'ensemble des articulations endommagées J est ajouté à l'entrée du réseau de neurones ("Est-il utile de savoir quelles sont les articulations endommagées ?") ;
— **Addition** $J + dq$: l'ensemble des articulations endommagées ainsi que le vecteur de vitesse des articulations du robot au moment de la détection sont ajoutés à l'entrée du réseau de neurones ("Est-ce que ces éléments sont utiles ?").

3.6 Résultat

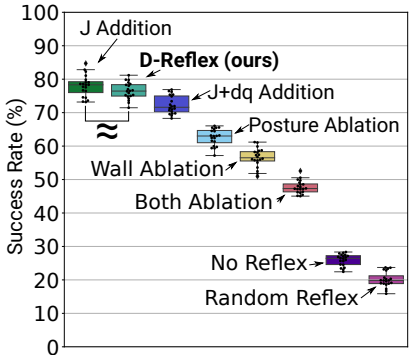

FIGURE 3.9 – Taux de succès en considérant uniquement les situations où la chute est évitable. Les boites à moustaches montre la médiane et les quartiles, les points sont les valeurs des différentes évaluations. Pour chaque variante, nous avons répliqué l'algorithme d'apprentissage 20 fois avec des répartitions du jeu de données différentes. Toutes les méthodes sont significativement différentes les unes des autres en utilisant un t-test avec une correction de Bonferroni (p-value $\leq 0,001$) sauf **D-Reflex** et **Addition** J.

La figure 3.7 présente des exemples de chutes évitées et de situations pour lesquelles la chute n'est pas évitable. La figure 3.9 présente une comparaison du taux de succès de notre méthode avec les différentes méthodes de référence, ablations, et additions. Toutes les méthodes sont significativement différentes les unes des autres sauf **Addition** J et notre méthode **D-Reflex**.

3.6.1 Comparaison principale

3.6.1.1 Notre méthode

D-Reflex permet au robot TALOS d'éviter la chute dans $76,4\% \pm 2,5\%$ des situations évitables (Figure 3.9). La figure 3.7 montre quelques exemples de rétablissement réussi ainsi que des exemples typiques d'échec. Les erreurs de notre méthode semblent le plus souvent être dues au fait que le réseau de neurones surestime la taille et la régularité des positions de contact (Figure 3.7, cinquième ligne). Une passe supplémentaire d'optimisation des hyperparamètres ou l'utilisation d'une méthode d'apprentissage supervisé plus performante pourrait permettre d'améliorer encore les performances.

Globalement, notre approche est la meilleure de notre comparaison (t-test avec une p-value $\leq 0,001$) pour éviter une chute.

3.6.1.2 Méthodes de références

Ne rien faire (**Sans Réflexe**) a un taux de succès faible ($26,4\% \pm 1,7\%$). Ce n'est pas un taux nul car une certaine partie des situations n'induisent pas de chute (par exemple si juste le mollet est bloqué).

Utiliser une position de contact aléatoire (**Réflexe aléatoire**) a un taux de succès inférieur ($19,8\% \pm 2,2\%$) car le réflexe induit un mouvement rapide et ample du bras qui, s'il n'est pas choisi correctement, induit une perte de stabilité et augmente les chances de chute.

Les performances de ces deux méthodes de référence montrent bien qu'il est pertinent d'apprendre un réflexe.

3.6.1.3 Ablations

Quand la position de contact est choisie indépendamment de la posture du robot et de la configuration du mur (**Ablation Totale**), le robot réussit à se stabiliser dans $47,3\% \pm 1,9\%$ des situations. Comparé à la performance de **D-Reflex** de $76,4\%$ cela montre que le réseau de neurones fait plus que choisir la meilleure solution en moyenne et que les points de contact dépendent de la situation.

Les performances des ablations séparées de la posture du robot (**Ablation de la Posture**) et de la configuration du mur (**Ablation du Mur**) montrent que les informations sur la posture du robot et la configuration sont deux informations nécessaires au choix de la position de contact, et que l'information sur la configuration est la plus importante (Figure 3.9).

3.6.1.4 Additions

Ajouter l'information sur quelles sont les articulations endommagées (**Addition J**) n'améliore pas significativement les performances et, comme cela ajoute une hypothèse forte sur nos connaissances de l'état du robot, cela confirme notre choix de ne pas supposer la connaissance de ces informations.

Ajouter en plus l'information sur la vitesse de chaque articulation (**Addition** $J + dq$) diminue les performances de façon significative. Comme le robot ne réalise pas de mouvements dynamiques pour réaliser sa mission, cette information ne semble pas utile pour déterminer la position de contact. Nous supposons que cette chute de performance est due à l'augmentation du vecteur d'entrée du réseau de neurones, et que nous pourrions obtenir des performances similaires à D-Reflex en réalisant une optimisation des hyperparamètres du réseau de neurones.

3.7 Expérimentations et analyses complémentaires

3.7.1 Robot réel

Nous avons choisi d'utiliser comme mur un matelas de chute sportif afin de minimiser les dommages du robot. Par simplicité, nous avons décidé d'utiliser comme dommage le genou passif, ce qui correspond sur le robot réel à couper l'alimentation de l'actionneur du genou.

Pour réaliser une preuve de concept sur le robot réel sans avoir à implémenter l'ensemble des détails que nous permet la simulation (la détection du mur et la détection du contact avec le mur), nous avons sélectionné en simulation 4 situations possédant un réflexe réussi et avons placé le robot à la même position par rapport au mur et dans la même posture.

Sur 4 situations ainsi testées, 3 des réflexes sur le robot réel ont permis au robot de retrouver la stabilité alors qu'il tombe sans ces réflexes. Ces tests montrent, d'une part, que le robot réel est suffisamment rapide pour réaliser les réflexes appris en simulation, et d'autre part, qu'il existe des solutions assez robustes pour passer de la simulation au monde réel.

3.7.2 Robustesse à la friction

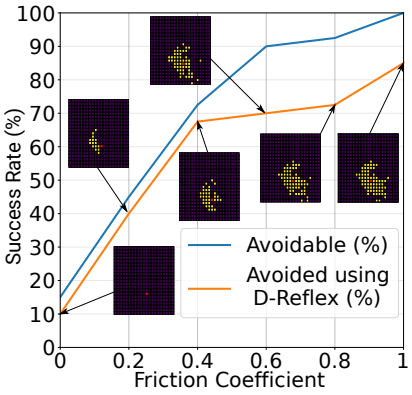

FIGURE 3.10 – Pourcentage de situations évitables et situations évitées à l'aide de D-Reflex, avec un réseau de neurones entraîné pour une friction de 1, en fonction de la friction sur le mur, ainsi que des exemples de la carte de contacts réelle.

Le simulateur physique DART simule le contact en suivant la dynamique de contact rigide, prenant en compte la friction, les rebonds et la restitution, chacun pouvant être paramétré pour simuler différents matériaux.

Dans nos expériences, la friction a sa valeur par défaut de 1 et la restitution est à 0. Nous avons évalué les performances d'un D-Reflex entraîné avec des données utilisant une friction de 1 sur différentes valeurs de frictions de 0 à 1 (Figure 3.10). Diminuer la friction a pour effet de diminuer le nombre de positions de contact réussies. Comme notre méthode choisit le point avec la plus forte estimation de succès et que ce point se trouve le plus souvent au centre de la région de succès, notre méthode est assez robuste à la diminution de la friction entre 1 et 0,4 avec une chute de performance de 85% à 67,5%. En dessous de 0,4 de friction, les performances chutent rapidement du fait que le mur devient glissant (Figure 3.10).

3.7.3 Utilisation du modèle endommagé dans le contrôleur corps complet

Nous avons évalué une autre méthode de référence correspondant au cas où le modèle utilisé par le contrôleur corps complet est celui du robot endommagé. Par simplicité, nous avons évalué dans différentes situations uniquement l'amputation du genou. Nous avons répliqué l'ensemble du processus d'apprentissage 25 fois (la seule différence étant donc que le contrôleur corps complet utilise le modèle réel du robot endommagé).

Les performances avec le modèle endommagé (taux de succès médian $82,3\% \pm 2,4\%$) sont statistiquement équivalentes à celles obtenues avec le modèle non endommagé ($80,7\% \pm 2,5\%$).

Cette faible différence de performance est due au fait que notre méthode ne prend pas en compte l'exactitude du mouvement par rapport à la commande mais uniquement le résultat de la simulation pour décider d'un succès ou non. Un contrôleur avec un modèle à jour a plus de chance de placer la main du robot proche de la position cible, tandis qu'un contrôleur avec un modèle incorrect aura tendance à avoir plus d'erreur de placement par rapport à la cible. Cela n'a pas d'importance car notre méthode ne stocke dans les cartes de contacts, que le succès ou non d'une position donnée en commande indépendamment de si cette position est atteinte ou non sur le mur. Autrement dit, le contrôleur corps complet est utilisé pour générer une politique et le simulateur informe l'algorithme de la qualité de cette politique.

De façon similaire, un algorithme de planification ou de commande prédictive (Dantec et al., 2021 ; Kurtz et al., 2021) pourrait être utilisé pour générer les trajectoires jouées dans notre simulateur. Dans tous les cas, il faut une méthode pour sélectionner la position de contact, ce que D-Reflex résout.

3.7.4 Prédiction de la chute

Comme il y a 25% des situations pour lesquelles le robot ne chute pas (Figure 3.9 *No Reflex*), nous nous sommes demandés s'il pouvait être intéressant de pouvoir classifier les situations provoquant la chute et celles ne provoquant pas la chute. Pour ce faire, nous avons échantillonné 10 000 situations différentes sans le mur, c'est-à-dire posture q du robot et combinaison de dommage à la jambe, et évalué en simulation si la situation entraînait la chute.

À l'aide de ce jeu de données, nous avons entraîné un réseau de neurones à classifier la chute ou non, en fonction de différents paramètres : la posture q, les articulations endommagées J et le type de dommage pour chacune (c'est-à-dire passif, bloqué ou amputé).

L'exactitude sur le *jeu de données d'entraînement* est la suivante :

— $48,6\%$ en connaissant la posture q ;
— $68,6\%$ en connaissant la posture q et les articulations endommagées J ;
— $93,9\%$ en connaissant la posture q, les articulations endommagées J ainsi que le type de dommage pour chacune.

Nous en avons tiré comme conclusion qu'il était nécessaire de connaître le type des dommages pour pouvoir prédire la chute. Cette hypothèse semble raisonnable étant donné qu'une amputation aura beaucoup plus d'impact qu'une articulation bloquée. Comme nous ne faisons pas l'hypothèse de savoir l'origine du dommage, le mieux que nous puissions faire est de trouver des réflexes robustes. Une alternative non étudiée ici pourrait être de détecter la chute à partir de données inertielles.

3.7.5 Robustesse d'une commande

La figure 3.11 présente pour une situation (c'est-à-dire posture du robot et configuration du mur) l'ensemble des positions de contacts atteintes pour chaque combinaison de dommages (Section 3.2.2) pour différentes commandes.

Selon la combinaison de dommages, une même commande peut atteindre des positions de contacts différentes sur le mur pouvant être éloignées de plusieurs dizaines de centimètres. Ceci s'explique par les différences de dynamique du robot endommagé qui chute dans différentes directions à différentes vitesses.

Notre méthode optimise pour une combinaison de dommages donnée les commandes qui permettent de rétablir la stabilité du robot. Ce choix est valide car en interrogeant le simulateur, notre méthode ne tient pas compte de la position de contact atteinte mais uniquement de la réussite ou non à rétablir la stabilité. De plus, les commandes qui contiennent des solutions sont regroupées ensemble, ce qui implique l'existence d'une commande robuste maximisant le nombre de réflexes réussis pour l'ensemble des combinaisons de dommages. Notre hypothèse sur la réussite de notre méthode réside dans le fait que le réseau de neurones classifieur apprend une représentation lisse des solutions qui permet de sélectionner une solution robuste se trouvant au centre d'un regroupement de solutions.

3.8 Conclusion

Étant donné la posture du robot et la configuration (distance et orientation) d'un mur à portée, notre méthode, D-Reflex, utilise un réseau de neurones classifieur pour estimer une position de contact qui permet à un robot humanoïde d'éviter la chute dans plus de 75% des cas où elle est évitable. D-Reflex est robuste à différentes combinaisons de dommages

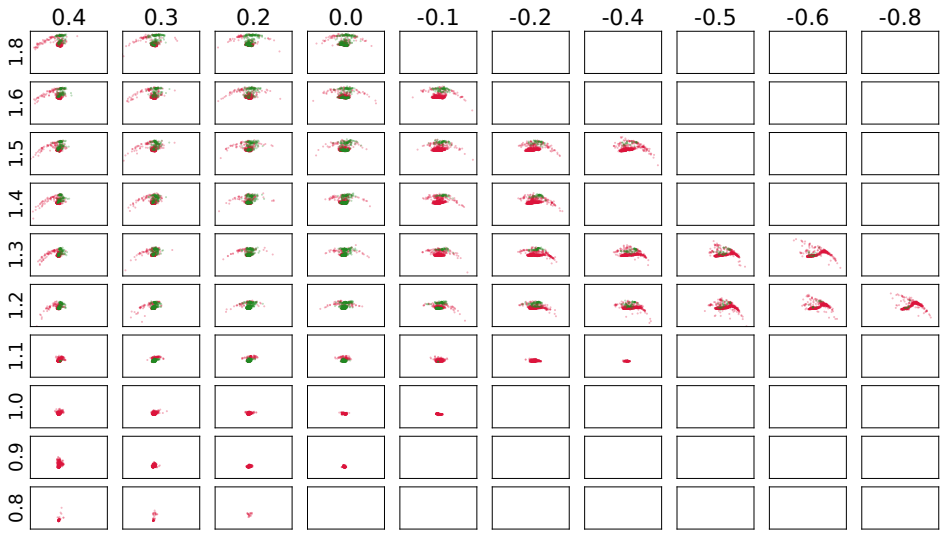

FIGURE 3.11 – Cette grille de boîtes présente pour une situation (c'est-à-dire posture du robot et configuration du mur) l'ensemble des positions de contacts atteintes pour chaque combinaison de dommages (Section 3.2.2) pour différentes commandes. Chaque boîte représente une commande et chaque point dans une boîte représente la position de contact atteinte sur le mur pour une combinaison de dommages (la taille de chaque boîte est fixée entre -1,5 m et 1,6 m sur l'axe horizontal et 0,3 m et 1,9 m sur l'axe vertical). Les points verts correspondent aux réflexes réussis et les points rouges aux réflexes échoués. Si la boîte est vide, c'est qu'aucune commande n'a permis d'entrer en contact avec le mur (car par exemple la commande demande de déplacer la main trop loin et le contrôleur corps complet renvoie une erreur). Selon la combinaison de dommages, une même commande peut atteindre des points différents pouvant être éloignés de plusieurs dizaines de centimètres.

(actionneur bloqué, actionneur passif et amputation) et ne requiert pas la connaissance du modèle endommagé du robot. Il utilise un contrôleur corps complet (Dalin et al., 2021) et des informations sensorielles facile à acquérir.

Une fois stabilisé, le robot aura besoin de mettre à jour son modèle et sa posture afin de pouvoir être contrôlé de nouveau. Pour ce faire, de nombreuses méthodes d'identification de modèle peuvent être étudiées (Hollerbach et al., 2008). Le robot peut ensuite continuer sa mission malgré ses dommages. L'hypothèse centrale de ce travail est qu'il existe un mur proche du robot.

D-Reflex est conçu pour éviter des chutes lors de missions complexes et risquées mais importantes avec un robot humanoïde. Dans de telles situations, il est attendu de l'opérateur de téléopérer le robot avec prudence en évitant des mouvements dynamiques et en le gardant en équilibre statique. Le développement de notre méthode pour des mouvements dynamiques comme une marche dynamique, des mouvements rapides ou sur un seul point de support (le pied) nécessite de résoudre plusieurs défis. D'une part, il faut réfléchir à une représentation des situations qui permette de prendre en compte la dynamique du robot, par exemple en prenant en compte la dynamique du centre de masse avec des méthodes similaires à l'étude du point de moment d'inclinaison nul (Sugihara et al., 2002). D'autre part, lors d'un mouvement dynamique, il y a une inertie supplémentaire à celle engendrée par la chute, ce qui demande une meilleure absorption de l'énergie pour permettre au robot de se stabiliser. Pour ce faire, nous pourrions coupler notre méthode de recherche de contact avec des méthodes d'optimisation

de l'absorption de l'énergie (Cui et al., 2021), de la posture (Samy et Kheddar, 2015), des gains de contrôle (Samy, Bouyarmane et al., 2017) ou de la distribution de l'impact (Samy, Caron et al., 2017).

Une limitation de l'étude actuelle de notre méthode est l'intégration pour des robots réels. D'une part, il faudrait intégrer l'ensemble des composants dans l'ordinateur de bord du robot et inclure la détection de surfaces suffisamment planes en temps réel en tant que tâche de fond du robot. D'autre part, il faudrait étudier la généralisation de notre méthode à d'autres robots humanoïdes. En particulier, pour savoir si le réflexe doit être réappris pour chaque morphologie ou si une forme d'apprentissage multi-tâche peut être utilisée pour les différentes morphologies.

Dans le chapitre suivant, nous allons étudier l'amélioration de la première phase de résolution pour pouvoir l'appliquer à des problèmes de plus hautes dimensions. Pour ce faire, nous appliquerons des méthodes multitâches qui permettent d'exploiter la similarité entre les différentes tâches pour améliorer l'efficacité d'échantillonnage et donc accélérer la résolution globale de tous les problèmes. Nous l'illustrerons en sélectionnant des positions de contact pour les deux mains et dans des environnements plus réalistes. Dans ces conditions, il ne sera plus possible d'utiliser une simple grille 2D pour couvrir les potentielles positions de contact : le principal défi est donc de construire le jeu de données de réflexes avec suffisamment de réflexes réussis.

Chapitre 4

Améliorer l'étape de résolution à l'aide de qualité-diversité multi-tâche

Contribution : Ce chapitre s'appuie sur le papier de poster *Multi-Task Multi-Behavior MAP-Elites* (MTMB-ME) (Anne et Mouret, 2023) pour lequel j'ai réalisé l'ensemble de l'étude sous la supervision de Jean-Baptiste Mouret (Section 4.2). Nous proposons ensuite deux extensions (Section 4.3) pour évaluer l'étape de généralisation : (1) une évaluation de l'étape de généralisation à partir d'un jeu de données produit par MTMB-ME comparé à PPO (Schulman, Wolski et al., 2017) et (2) l'utilisation de la méthode sur un environnement plus réaliste. Nous avons trouvé ces deux résultats peu impactants et avons décidé de ne pas proposer de publication.

Sommaire

4.1 Introduction

Dans le chapitre précédent, nous avons découpé le problème d'apprentissage en une étape de résolution d'un ensemble de situations (posture du robot, configuration du mur et dommages de la jambe) et une étape de généralisation à d'autres situations via de l'apprentissage supervisé. Lors de la première étape de résolution, nous avons utilisé une simple recherche en grille appliquée à chaque situation pour obtenir un ensemble de solutions réussies et ratées

pour chaque situation (sous forme d'une carte de positions de contact sur le mur). Ces solutions différentes sont ensuite utilisées pour déterminer une solution robuste à l'aide de l'étape de généralisation.

Nous pouvons déjà généraliser le principe de situation avec le principe de tâche. Dans le chapitre précédent, chaque tâche est ainsi définie par une situation (posture du robot, configuration du mur et dommages de la jambe), ce qui caractérise la simulation et donc la *fitness* (le fait que le robot soit toujours stable au bout de 15 s de simulation). Dans le cas général, une tâche correspond à une fonction de *fitness* qu'il faut maximiser.

Nous pouvons donc formaliser le problème résolu lors du chapitre précédent. Pour un ensemble de n tâches $T_{1:n}$ partageant le même espace de commande $\mathcal{C}$, mais où chaque tâche T_i est définie par une fonction de *fitness* différente f_i, nous voulions trouver une commande robuste c_i^* pour chaque tâche T_i. Pour ce faire, la recherche par grille teste un ensemble de m_i commandes $c_i^{1:m_i}$ pour chaque tâche T_i, ce qui permet de trouver un ensemble de commandes réussies $C_{\text{réussies}} = \left\{c_i^j | f_i(c_i^j) = f_{max}\right\}$ (que nous appellerons des solutions) et de commandes ratées $C_{\text{ratées}} = \left\{c_i^j | f_i(c_i^j) < f_{max}\right\}$ (pour le problème du réflexe du chapitre précédent, la valeur de *fitness* est binaire). L'étape de généralisation apprend ensuite un classifieur pour distinguer les commandes réussies $C_{\text{réussies}}$ des commandes ratées $C_{\text{ratées}}$. Lors de l'inférence, ce classifieur est utilisé pour sélectionner une commande robuste c^* pour la tâche courante en sélectionnant empiriquement une solution se trouvant au centre des autres solutions.

Le problème est qu'une recherche en grille ne passe pas à l'échelle pour plus de 2 ou 3 dimensions. Par exemple, pour le problème d'apprentissage d'un réflexe d'évitement de chute avec les deux mains d'un robot humanoïde, il faudrait, pour obtenir la même résolution que dans le chapitre précédent, effectuer plus de $200,000$ simulations par tâche. Il nous faut donc une méthode plus efficace en échantillonnage. Nous pouvons remarquer que l'objectif est uniquement de trouver, à la fin, une commande robuste c_i^* pour chaque tâche T_i. En particulier, l'ensemble des commandes ratées $C_{\text{ratées}}$ ne nous intéresse pas alors que, pour les problèmes épars, cet ensemble comprend la majorité des évaluations effectuées. Nous pouvons donc reformuler le problème comme un problème d'optimisation multi-tâche, c'est-à-dire de trouver pour un ensemble de n tâches la solution de fitness maximale pour chaque tâche :

$$c_i^* = \underset{c \in \mathcal{C}}{\operatorname{argmax}} f_i(c), 1 \leq i \leq n.$$

Dans ce chapitre, notre idée principale est que ce type de problème peut être résolu avec un algorithme d'optimisation multi-tâche. Au lieu de résoudre chaque tâche indépendamment, l'optimisation multi-tâche (Gupta, Y.-S. Ong et al., 2016 ; Mouret et Maguire, 2020) exploite la similarité entre les différentes tâches pour améliorer l'efficacité d'échantillonnage. *Multi-Task MAP-Elites* (MT-ME) (Mouret et Maguire, 2020) est un algorithme d'optimisation multi-tâche boîte noire qui utilise les mêmes idées que MAP-Elites (Mouret et Clune, 2015) (Section 2.5.1.2) pour résoudre un ensemble de tâches en exploitant une archive de solutions pour trouver une solution performante pour chaque tâche.

Un choix important que nous faisons est que nous ne voulons pas optimiser directement la robustesse, ce qui pourrait être fait par exemple en évaluant l'effort du robot, les forces d'impact du réflexe, ou en réévaluant chaque solution avec du bruit. À la place, nous voulons estimer les solutions robustes en utilisant le même principe que le chapitre précédent, c'est-à-dire avec une diversité de solutions pour sélectionner une solution dont les solutions voisines sont aussi de bonnes solutions.

Plus formellement, nous voulons une méthode qui, pour un ensemble de tâches $T_{1:n}$, trouve m_i solutions $c_i^{1:m_i}$ différentes pour chaque tâche T_i lors de la phase de résolution (dans l'idée que la phase de généralisation s'occupera d'apprendre les solutions robustes grâce à un classifieur). Pour une tâche donnée, ce problème correspond au problème de qualité-diversité que résout MAP-Elites (Mouret et Clune, 2015). La différence est que nous ne voulons pas trouver une diversité de solutions pour une tâche mais pour un ensemble de tâches. Ce problème correspond donc au problème de qualité-diversité multi-tâche, et à notre connaissance, il n'existe aucun algorithme qui le résout.

Dans ce chapitre, nous présentons donc une nouvelle méthode de qualité-diversité multi-tâche, que nous appelons *Multi-Task Multi-Behavior MAP-Elites* (MTMB-ME), qui combine MAP-Elites et MT-ME. Étant donné un ensemble de tâches, MTMB-MAP Elites construit une archive de solutions diverses et performantes pour chaque tâche.

Comme le chapitre précédent, nous sommes particulièrement intéressés par son utilisation pour construire une archive de réflexes pour un robot humanoïde. Pour ce problème, une tâche correspond à un placement spécifique du mur, à une posture du robot et à une combinaison de dommages à la jambe. Une solution correspond à la position de contact d'une ou des deux mains sur le mur. La principale différence du problème avec le chapitre précédent est l'utilisation des deux mains, ce qui fait passer l'espace de recherche de 2 à 4 dimensions. La figure 4.1 montre un exemple d'archive collectée à l'aide de notre méthode avec des instantanés du robot pour les solutions trouvées sur deux situations.

Ce chapitre étudie d'abord la première phase de résolution (Section 4.2) puis la seconde phase de généralisation des solutions (Section 4.3.1). Une première étude sur la généralisation du réflexe à deux bras sur mur parfait montre que le robot n'a pas besoin des deux bras pour la famille de chutes engendrées par les dommages appliqués (Section 4.3.1.1). La généralisation de la nouvelle méthode de résolution avec un seul bras est comparée avec PPO (Schulman, Wolski et al., 2017) sur un mur parfait (Section 4.3.1.2). Enfin, le chapitre présente comment étendre la méthode à des environnements plus réalistes provenant d'une numérisation 3D d'un environnement réel (Section 4.3.2).

4.2 *Multi-Task Multi-Behavior MAP-Elites*

4.2.1 *Multi-Task MAP-Elites*

MT-ME (Mouret et Maguire, 2020) résout le problème d'optimisation multi-tâche jusqu'à plusieurs milliers de tâches simultanément. Dans une illustration en robotique pour apprendre des marches pour différentes morphologies de robot, au lieu d'optimiser chaque morphologie individuellement, MT-ME recherche les marches optimales pour 2 000 morphologies en même temps. Les marches trouvées permettent d'atteindre des distances trois fois supérieures à celles trouvées par CMA-ES (Hansen et al., 2003) exécuté individuellement sur chaque tâche avec le même nombre d'évaluations globales. Ceci est dû au fait que CMA-ES repart de zéro pour chaque tâche alors que MT-ME exploite les solutions trouvées pour une tâche pour résoudre une autre tâche. MT-ME a aussi d'autres applications, par exemple dans des scénarios industriels pour optimiser des comportements ergonomiques pour des travailleurs de différente morphologie à leur poste de travail (Zhong, Weistroffer et al., 2023).

MT-ME commence par discrétiser explicitement l'espace des tâches en un nombre fini de tâches et en mettant à jour pour chaque tâche la meilleure solution connue, appelée une élite, sous la forme d'une archive d'élites. À chaque itération, il sélectionne aléatoirement deux élites de l'archive, génère une solution candidate en utilisant un opérateur de variation

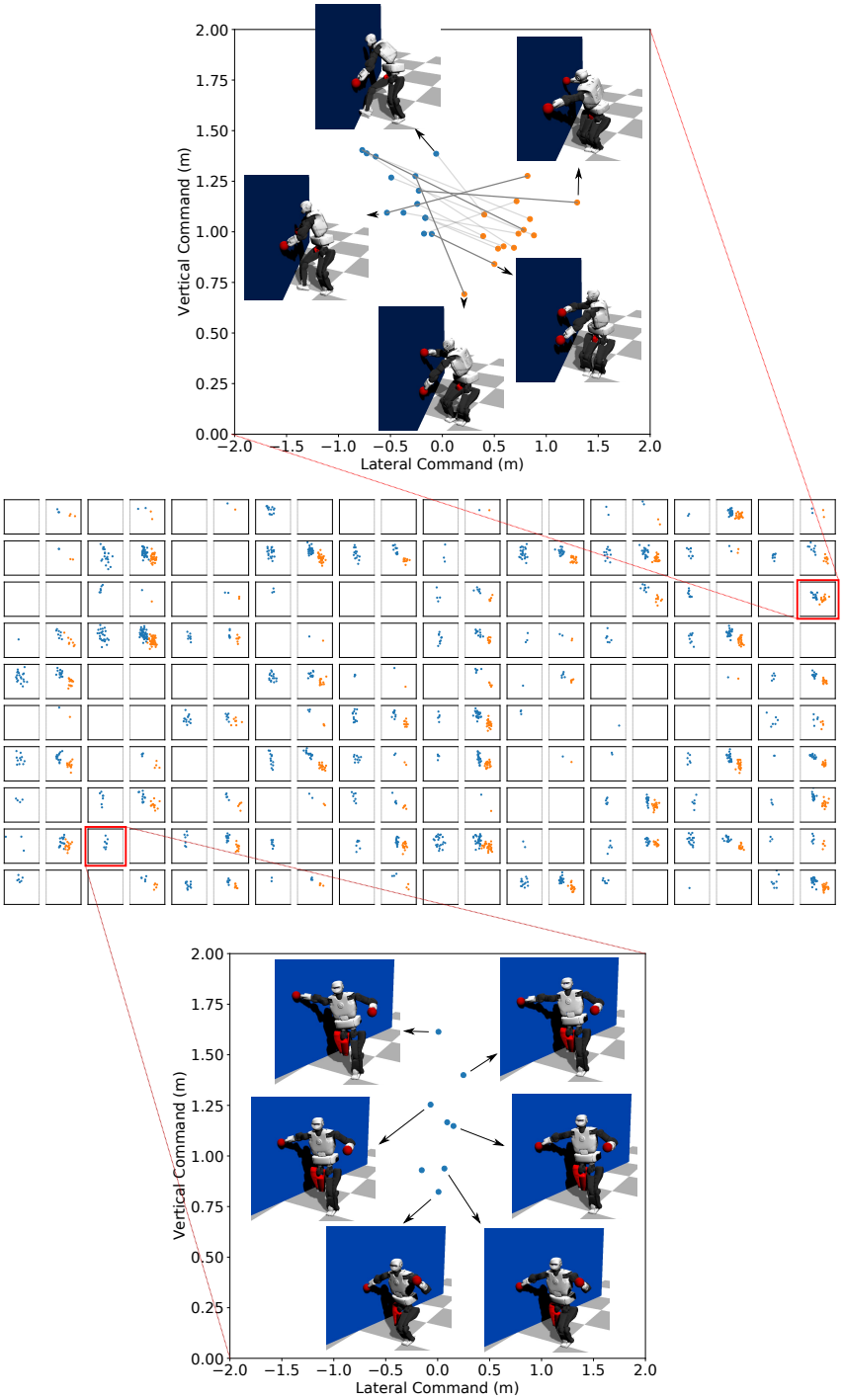

FIGURE 4.1 – Archive collectée sur 200 tâches (100 situations avec (1) la main droite et (2) les deux mains) en utilisant MTMB-MAP-Elites sur un ensemble de tâches d'un robot humanoïde pour se rétablir après un dommage. Chaque situation correspond à la posture du robot humanoïde, une combinaison de dommages à sa jambe qui a de fortes chances de le faire tomber et la distance et l'orientation d'un mur à portée de bras. Le but est de trouver autant de positions de contact sur le mur que possible. Chaque paire de boîtes montre les solutions trouvées pour une situation, séparées en deux tâches : utilisant uniquement la main droite (en bleu) et utilisant les deux mains (en bleu et orange). Des instantanés des solutions trouvées sont aussi présentés pour deux tâches : utilisant la main droite (à gauche) et utilisant les deux mains (à droite).

(comme *iso-line* (Vassiliades et Mouret, 2018) ou SBX (Agrawal et al., 2000)) sur les deux élites, et évalue la solution candidate sur une des tâches tirée aléatoirement. Si la tâche n'avait pas de solution connue ou si la nouvelle solution a une *fitness* plus élevée que l'élite courante, la solution candidate devient l'élite pour cette tâche, et l'algorithme recommence sa boucle principale. La différence avec MAP-Elites est que l'espace des tâches remplace l'espace des comportements. Cela implique en particulier que la cellule de l'archive est connue avant d'évaluer la solution car il faut choisir la tâche d'évaluation pour pouvoir évaluer (à l'opposé du comportement qui est un résultat de l'évaluation).

Dans ce chapitre nous allons modifier l'algorithme de MT-ME avec l'algorithme de MAP-Elites pour résoudre le problème de qualité-diversité multi-tâche.

4.2.2 Formulation du problème de qualité-diversité multi-tâche

Étant donné un ensemble de tâches $T_{1:n} \in \mathcal{T}$, le but est de trouver pour chaque tâche T_i autant de solutions différentes $(s_i^j)_{j \leq m_i}$ qu'il est possible où m_i correspond au nombre de solutions différentes trouvées pour la tâche T_i.

L'espace des tâches $\mathcal{T}$ définit un espace de commandes $\mathcal{C}$, un espace des comportements $\mathcal{B}$, une fonction de $fitness : \mathcal{T} \times \mathcal{C} \to \mathbb{R}$ et une fonction de description comportementale $behavior : \mathcal{T} \times \mathcal{C} \to \mathcal{B}$. Nous faisons aussi l'hypothèse que la fonction de *fitness* est bornée par $f_{max} \in \mathbb{R}$ de telle façon qu'une commande c peut être dite optimale pour la tâche T si et seulement si $fitness(T, c) = f_{max}$.

Plus formellement, la formulation du problème est la suivante :

$$\begin{aligned} &\text{maximiser } \textstyle\sum_{i=1}^{n} m_i \\ &\text{tel que } fitness(T_i, c_i^j) = f_{max} \ \forall i, \forall j \leq m_i \\ &\text{tel que } behavior(T_i, c_i^j) \neq behavior(T_i, c_i^k) \ \forall i, \forall k \neq j \end{aligned} \tag{4.1}$$

4.2.3 *Multi-Task Multi-Behavior MAP-Elites*

Pour résoudre ce problème, nous proposons MTMB-MAP-Elites comme combinaison de MAP-Elites et MT-ME. La principale différence est que nous ne cherchons plus la solution la plus performante pour chaque tâche, mais que nous cherchons le plus de solutions différentes pour chaque tâche. Nous faisons l'hypothèse que les différentes tâches ont suffisamment de similarités pour partager une partie des solutions. Ce qui justifie le fait de les résoudre en même temps pour économiser du temps et du calcul par rapport à simplement utiliser MAP-Elites (Mouret et Clune, 2015) sur chacune des tâches séparément.

Par clarté, dans le reste de ce chapitre, nous appelons "élite" une commande c qui a été évaluée et stockée dans l'archive, et "solution" une élite optimale, c'est-à-dire une élite avec une *fitness* maximale f_{max}.

L'algorithme est le suivant :

1. Initialisation :
   (a) sélectionner n tâches aléatoires $T_{1:n} \in \mathcal{T}$;
   (b) choisir un budget B d'évaluations (B doit être significativement plus grand que n) ;
   (c) initialiser l'archive avec des commandes aléatoires sur des tâches sélectionnées aléatoirement jusqu'à obtenir un nombre suffisant d'élites (par défaut, l'algorithme arrête l'initialisation quand il a trouvé n élites).
2. Boucle principale de l'algorithme répétée $B - b_{init}$ fois où b_{init} est le nombre d'évaluations effectuées lors de l'initialisation :

(a) sélectionner une commande c en choisissant aléatoirement deux tâches T_i et T_j avec des élites, puis sélectionner aléatoirement une élite pour chacune c_i et c_j et appliquer un opérateur de variation ;
(b) sélectionner une tâche T_k au hasard ;
(c) évaluer la commande c en collectant le comportement $b_k = behavior(T_k, c)$ et la *fitness* correspondante $f = fitness(T_k, c)$;
(d) mettre à jour l'archive : si le comportement b_k n'est pas présent, il est ajouté à la tâche T_k, et s'il est déjà présent et que la nouvelle *fitness* f est strictement plus grande que l'ancienne, l'ancienne élite est remplacée par la nouvelle.

Dans ce chapitre, nous utilisons comme opérateur de variation une variante de *iso-line* (Vassiliades et Mouret, 2018). Plus formellement, étant donné deux parents c_i et c_j,

$$c = \alpha c_i + (1 - \alpha)c_j + \mathcal{N}(0, \sigma)$$

où α est échantillonné uniformément dans $[0, 1]$. Nous avons choisi cet opérateur car celui-ci utilise un croisement (en plus d'une mutation via un bruit gaussien). Dans le cas d'un algorithme élitiste utilisant une archive, l'utilisation d'un croisement a montré de bonnes performances pour explorer l'espace des solutions en biaisant les nouvelles solutions vers le sous-espace des "bonnes solutions", appelé l'hyper-volume des élites.

4.2.4 Expérimentation

MTMB-MAP-Elites est évalué sur un ensemble de tâches de rétablissement de l'équilibre après dommages pour le robot humanoïde TALOS en simulation. La différence avec le chapitre précédent est que cette fois-ci le robot peut soit utiliser sa main droite, soit utiliser ses deux mains (l'utilisation de la main gauche uniquement n'est pas pertinente, car dans la plupart des cas, le robot tombe sur la droite).

4.2.4.1 L'espace des tâches

L'espace des tâches $\mathcal{T}$ est défini comme l'ensemble des différentes situations correspondant à la posture du robot q, la configuration du mur (distance d et orientation α) et la combinaison de dommages sur la jambe.

Pour sélectionner n tâches aléatoires $T_{1:n}$, l'algorithme échantillonne de façon identique au chapitre précédent différentes situations avec rejet en cas de contact avant le mur (Section 3.4.1.2).

4.2.4.2 L'espace des commandes

Tout comme dans le chapitre précédent, la commande correspond à la position de contact "cible" sur le mur qui est envoyée au contrôleur corps complet en tant que tâche de contact avec un poids élevé.

La diversité est favorisée en dupliquant explicitement chaque situation en deux tâches : une tâche qui utilise uniquement la main droite et une tâche qui utilise les deux mains. L'espace des commandes est défini comme $\mathcal{C} = X \times Z \times X \times Z$ où l'axe latéral $X = [x_{min}, x_{max}]$ et l'axe vertical $Z = [z_{min}, z_{max}]$ sont choisis pour représenter grossièrement les positions de contacts accessibles sur le mur. Pour les tâches n'utilisant que la main droite, uniquement les deux premières dimensions sont utilisées.

4.2.4.3 L'espace des comportements

L'espace des comportements $\mathcal{B}$ est défini comme l'ensemble des positions cartésiennes "atteintes" par les mains sur le mur (en 2D pour la main droite et en 4D pour les deux mains). Nous différencions les positions de contact "cibles" et "atteintes", car la position atteinte est rarement la même que la position ciblée (Section 3.7.5).

Pour différencier les comportements, l'algorithme discrétise l'espace des comportements avec une grille de résolution de 20 cm et définit comme descripteur de comportement $b = behavior(T, c)$ l'indice dans la grille (ou de la paire d'indices pour les deux mains) contenant la position de contact "atteinte". Comme cet ensemble de comportements est vaste et que nombre d'entre eux ne sont pas réalisables (par exemple pour des raisons de portée ou d'auto-collision), l'algorithme initialise l'archive comme étant vide et ajoute de nouveaux comportements au fur et à mesure qu'ils sont découverts.

4.2.4.4 La fonction de *fitness*

La *fitness* correspond au temps de l'épisode avant que la simulation ne s'arrête pour cause d'auto-collision, de chute (contact non planifié entre le robot et le mur ou le sol), ou de la fin planifiée de 10 s, ainsi $f_{max} = 10$ s. Cette durée seuil de fin a été sélectionnée comme étant un bon compromis entre réduire le temps de simulation et diminuer le nombre de faux positifs (quand le robot ne tombe pas avant la fin mais serait tombé plus tard). Une expérience préliminaire nous a permis de déterminer que réduire le seuil de 15 s comme au chapitre précédent à 10 s n'engendre qu'un seul faux positif sur 310 situations.

4.2.5 Évaluation

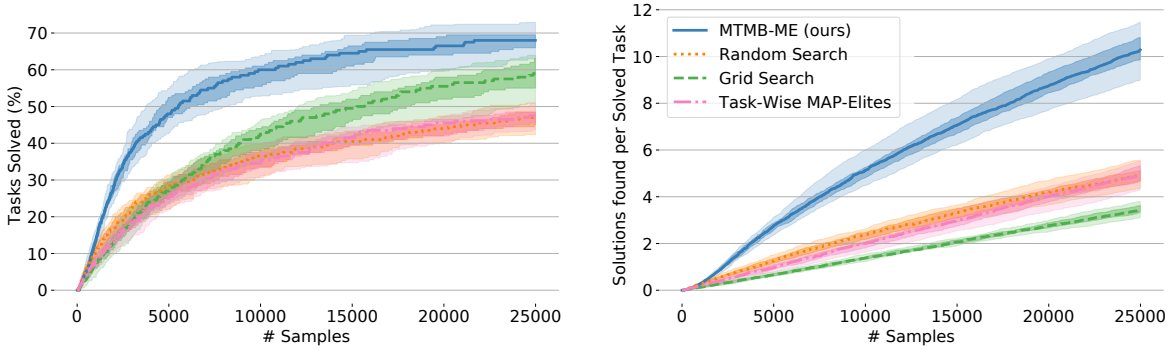

FIGURE 4.2 – Comparaison pour résoudre 200 tâches (100 situations avec soit la main droite soit les deux mains) entre notre méthode *MTMB-ME* et trois méthodes de référence : une recherche aléatoire **Random Search**, une recherche par grille **Grid Search** et MAP-Elites exécuté sur chaque tâche individuellement **Task-Wise MAP-Elites**. (a) Le pourcentage de tâches résolues et (b) le nombre de solutions par tâche résolue. La ligne pleine correspond à la médiane, la zone ombrée plus sombre aux quantiles [25%, 75%] des données et la zone ombrée plus claire aux quantiles [5%, 95%] sur 25 réplications. MTMB-MAP-Elites résout plus de tâches que la meilleure méthode de référence (+20, 8% que **Random Search**, +9, 9% que **Grid Search**, et +20, 7% que **Task-Wise MAP-Elites**) et plus important encore, trouve plus de deux fois plus de solutions par tâche résolue.

Pour l'évaluation, nous échantillonnons 100 situations en dupliquant chacune en deux tâches (la main droite et les deux mains), ce qui nous amène à $n = 200$ tâches. Le budget est fixé à $B = 25\,000$ simulations, en stoppant l'initialisation quand l'archive a collecté 200 élites.

À notre connaissance, il n'existe pas de méthode dans la littérature qui résout le problème de qualité-diversité multi-tâche. **MTMB-ME** est donc comparée à trois méthodes de référence assez naïves, en fixant à chaque fois le budget à $B = 25\,000$ simulations :

— **Random Search** : sélectionner la commande en échantillonnant selon une distribution uniforme sur $\mathcal{C}$.
— **Grid Search** : sélectionner les mêmes commandes pour chaque tâche réparties équitablement sur l'espace de commande $\mathcal{C}$ (pour avoir le même nombre d'évaluations par tâche, c'est-à-dire $\frac{B}{n} = 125$, nous utilisons une grille 5×5 pour la main droite et une grille 5×2 pour les deux mains) ;
— **Task-Wise MAP-Elites** : lancer MAP-Elites sur chaque tâche indépendamment avec un budget de 125 simulations par tâche.

Pour homogénéiser l'ordre de résolution de chaque tâche entre **MTMB-ME** et les méthodes de référence, nous avons échantillonné un ordre aléatoire dans lequel les évaluations sont comptabilisées de façon à ce que chaque tâche soit visitée de façon uniforme dans le temps.

Nous effectuons 25 réplications et évaluons deux critères :

— le pourcentage de tâches résolues (c'est-à-dire de tâches avec au moins une solution) ;
— le nombre de solutions par tâche résolue.

4.2.6 Résultat

La figure 4.1 montre l'exemple d'une archive collectée avec **MTMB-ME** pour l'une de ces replications. La figure 4.2 montre la comparaison de notre méthode **MTMB-ME** avec les trois méthodes de référence sur 25 réplications.

MTMB-ME obtient une meilleure performance que toutes les méthodes de référence en résolvant en moyenne $67,8\% \pm 3,7\%$ des tâches contre $47,0\% \pm 2,8\%$ pour **Random Search**, $57,9\% \pm 4,3\%$ pour **Grid Search**, et $47,1\% \pm 2,6\%$ pour **Task-Wise MAP-Elites**.

Plus important encore pour notre but de construire un jeu de données diversifié, **MTMB-ME** trouve en moyenne $10,2 \pm 0,8$ solutions par tâche résolue contre $4,9 \pm 0,4$ pour **Random Search**, $3,4 \pm 0,3$ pour **Grid Search**, $4,9 \pm 0,4$ pour **Task-Wise MAP-Elites**.

Avec un budget de seulement 125 évaluations par tâche, **Task-Wise MAP-Elites** réalise seulement une recherche aléatoire puis exploite rapidement les quelques élites qu'il a pu trouver, faisant ainsi de l'exploitation trop rapide et aboutissant à des performances similaires à celles de **Random Search**.

Grid Search recouvre l'espace de commande uniformément, lui permettant de résoudre presque autant de tâches que notre méthode mais réduisant fortement sa capacité à découvrir des solutions diverses. En combinant deux commandes provenant de deux élites différentes, **MTMB-ME** échantillonne indirectement à partir d'une distribution qui représente mieux le sous-espace des solutions possibles. Ceci lui permet d'accélérer l'exploration en rejetant les régions qui ont peu de chances de contenir des solutions.

4.2.7 Conclusion

MTMB-ME est un nouvel algorithme de qualité-diversité pour les problèmes multi-tâches en combinant MAP-Elites et MT-ME. MTMB-ME permet de construire un jeu de données de solutions diversifiées pour un ensemble de tâches de façon plus performante qu'en utilisant des méthodes individuellement sur chaque tâche. La section suivante évalue l'étape de généralisation à partir de ce jeu de données.

4.3 Extensions pour évaluer la généralisation

Une première extension du papier de poster étudie l'utilisation des deux mains pour apprendre un réflexe de prévention de chute sur un mur parfait (Section 4.3.1). Nous nous sommes d'abord rendu compte que même si la méthode permet de trouver des solutions qui utilisent les deux mains, ces solutions sont peu nécessaires. Autrement dit, un réflexe avec une seule main suffit la majorité du temps (Section 4.3.1.1). Nous ne comparons donc notre méthode de "résolution puis généralisation" à une méthode d'apprentissage par renforcement profond PPO que pour des réflexes qui utilisent une seule main (Section 4.3.1.2).

Une seconde extension étend notre méthode à un environnement plus réaliste en simulation (Section 4.3.2). Elle met en lumière le problème de robustesse du réflexe hors du mur parfait (Section 4.3.2.2), puis décrit les changements qu'apportent un environnement plus réaliste (Section 4.3.2.1). Elle détaille ensuite une piste d'amélioration de la robustesse qui n'a pas été concluante (Section 4.3.2.3). Enfin, elle compare la généralisation de notre méthode à un environnement plus réaliste contre une ablation qui utilise une première étape de résolution aléatoire (Section 4.3.2.4).

4.3.1 Généralisation avec mur parfait

4.3.1.1 Le robot n'a besoin que d'une main

Comparaison préliminaire avec PPO. De façon similaire au chapitre précédent, un unique classifieur est entraîné à l'aide du jeu de données créé par MTMB-ME pour déterminer si une commande avec une ou deux mains permet la stabilisation du robot après un dysfonctionnement. Comme méthode de référence, nous avons choisi d'utiliser PPO de par sa maturité et facilité d'usage (Section 2.3.2.1). L'espace des actions prises par l'agent est l'espace des commandes sélectionnées par MTMB-ME et le classifieur. L'espace des états est la représentation du mur : distance et orientation. Nous pouvons noter que comme l'épisode est d'horizon 1, l'espace des états se restreint à l'état initial. La fonction de récompense est la même fonction de *fitness*. Nous n'avons pas comparé avec une politique séquentielle car le contrôle d'un robot humanoïde est complexe. À notre connaissance, l'utilisation de l'apprentissage par renforcement profond pour réaliser du contrôle corps complet d'un robot humanoïde est toujours un problème non résolu.

Nous avons d'abord comparé plusieurs choix de commandes : ne rien faire, utiliser la main gauche, utiliser la main droite ou utiliser les deux mains. Avec ces quatre choix, PPO décide toujours de ne rien faire car c'est un minimum local de la fonction de récompense. D'une part, une partie des conditions ($25,3\%$) ne font pas tomber le robot. D'autre part, les capacités exploratoires de PPO sont principalement du bruit sur les actions, or réaliser un réflexe aléatoire augmente les chances de tomber (Section 3.6.1.2).

Nous avons ensuite réduit les choix en enlevant : (1) "la main gauche uniquement" car elle n'est pas utile et (2) "ne rien faire" car il est difficile de prédire si un réflexe est nécessaire avec uniquement pour information la posture courante (Section 3.7.4). Nous avons donc testé avec uniquement le choix entre utiliser la main droite ou utiliser les deux mains. Le tableau 4.1 montre que notre méthode et PPO ont un taux de succès similaire mais que PPO utilise uniquement la main droite. Ceci nous a amené à nous poser la question de l'utilité des deux mains pour notre problème.

TABLE 4.1 – Comparaison de l'évaluation d'une exécution des différentes méthodes pour éviter la chute avec le choix entre utiliser uniquement la main droite ou utiliser les deux mains.

	Taux de succès	Main droite	Deux mains
Ne rien Faire	25,3%		
Notre méthode	31,9%	60,6%	39,4%
PPO	31,1%	100%	0%
Notre méthode (main droite)	21,9%	100%	0%
Aléatoire	12,6%	51,3%	48,7%

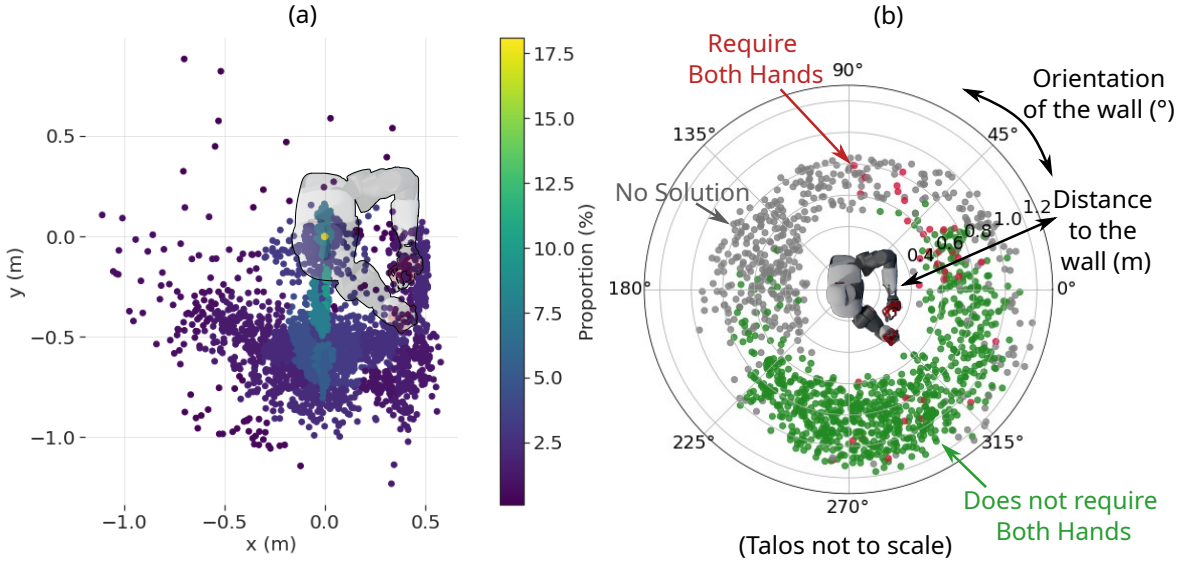

FIGURE 4.3 – (a) Nuage de points des positions du centre de masse de robot Talos après 10 000 dommages aléatoires. Le gradient de couleur représente la proportion de chute dans chaque région. Le robot Talos est ajouté en fond pour aider à la visualisation de la chute. (b) Nuage de points des configurations des murs autour du Talos pour lesquels MTMB-ME a trouvé des solutions : vert = le mur ne nécessite pas les deux mains (car il y a des solutions avec la main droite), rouge = le mur nécessite les deux mains et gris = pas de solution.

Est-ce que les deux mains sont utiles ? Sur les 25 jeux de données créés par MTMB-ME (Section 4.2.5), c'est-à-dire 2500 situations différentes, 18% d'entre elles n'ont pas de solution, 12% ont des solutions avec seulement les deux mains, 17% ont des solutions avec seulement la main droite et 53% ont des solutions avec les deux mains et avec la main droite. La figure 4.3.b montre dans l'espace d'orientation et position du mur autour du robot les types de solutions trouvées par MTMB-ME. Les deux mains servent principalement pour des murs qui se trouvent en face et à gauche du robot. Et surtout le robot ne peut éviter des chutes majoritairement que si le mur est à droite (du côté de sa jambe endommagée). C'est pourquoi utiliser les deux mains n'est utile que pour une faible partie des situations.

Une explication est que le robot tombe majoritairement à droite. La figure 4.3.a montre la position du centre de masse du robot lorsque nous appliquons un dommage aléatoire à sa jambe droite (c'est-à-dire avec la même distribution de dommages utilisée dans les différentes expériences). Nous pouvons voir qu'une bonne partie des dommages ne fait pas chuter le robot (c'est-à-dire le centre de masse n'a pas bougé) et que s'il chute, il chute principalement sur le côté droit.

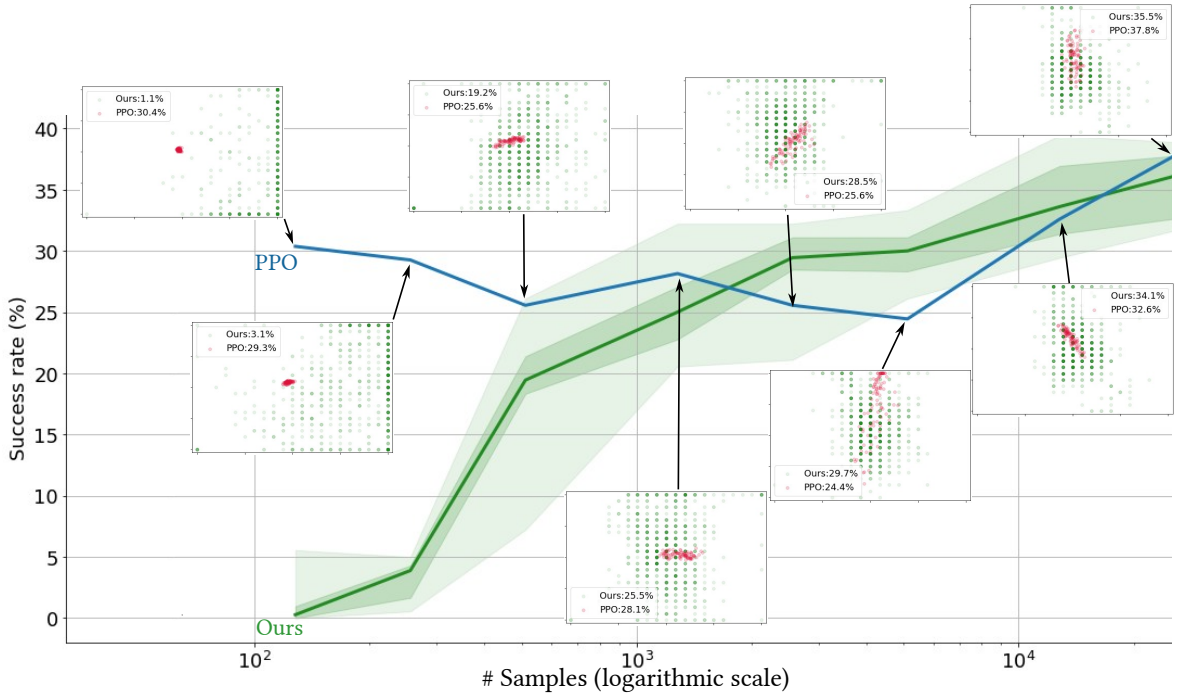

FIGURE 4.4 – Comparaison du taux de succès de validation lors de l'entraînement avec une récompense éparse (1 si le robot est toujours debout au bout de 10 s et 0 sinon) et un espace de solutions centré sur de bonnes solutions (le centre de l'espace correspond à cibler le centre du mur, ce qui est un bon premier réflexe) d'une exécution de PPO et de plusieurs exécutions de notre méthode (pour obtenir un score, plusieurs classifieurs sont entraînés au fur et à mesure que MTMB-ME remplit l'archive). Les nuages de points correspondent aux solutions sélectionnées par PPO (en rouge) et notre méthode *Ours* (en vert) lors d'évaluations de validation avec en label le taux de succès correspondant.

Il semble donc logique qu'un mur à gauche ne puisse aider et qu'au contraire un mur à droite soit le meilleur support possible. De ce fait, utiliser les deux mains n'aide pas beaucoup le robot. D'une part, pour pouvoir placer sa main gauche sur un mur à droite, le robot doit tourner son buste, ce qui entraîne un mouvement dynamique que le contrôleur corps complet peut refuser d'appliquer lorsque le mouvement est trop important et qui a de fortes chances de déstabiliser le robot davantage. D'autre part, un double appui sur un mur frontal ne permet pas facilement au contrôleur corps complet de stabiliser le robot. La raison est que le robot utilise un contrôle en position, ce qui le rend rigide : au mieux, le robot a suffisamment de frottement avec ses deux mains pour l'empêcher de glisser sur le côté de sa jambe endommagée, au pire le robot rebondit et perd le contact avec le mur.

4.3.1.2 Comparaison avec PPO avec une seule main

Ayant exclu l'utilisation des deux mains, le robot revient à l'utilisation uniquement de sa main droite comme au chapitre précédent. La différence réside dans l'utilisation d'une méthode de qualité-diversité multi-tâche, MTMB-ME, pour trouver plus de solutions par rapport à réaliser une recherche en grille, et donc de pouvoir visiter plus de situations différentes pour un même budget d'évaluations.

Cette section compare notre méthode à PPO avec uniquement la main droite. Nous avons d'abord testé différentes fonctions de *fitness* (correspondant à la fonction de récompense).

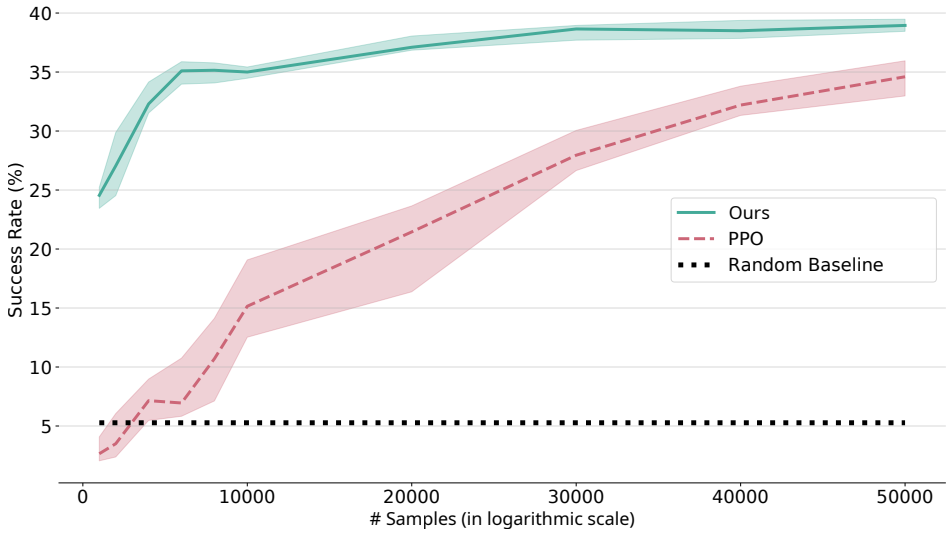

FIGURE 4.5 – Comparaison de la généralisation de notre méthode (MTMB-ME puis généralisation avec un classifieur) et PPO avec une méthode de référence utilisant un réflexe aléatoire. Le trait représente la médiane et la zone colorée représente le premier et troisième quartile de 10 réplications sur un jeu de données d'évaluations de 1000 situations échantillonnées aléatoirement.

Une récompense éparse Avec une récompense booléenne indiquant uniquement la stabilisation du robot au bout de 10 s, PPO n'y arrive que si l'espace des actions est centré sur (0,0). En effet, viser le centre du mur est un bon choix initial qu'il peut ensuite optimiser. La figure 4.4 montre qu'avec ces poids initiaux, PPO se concentre au centre de l'espace des solutions et obtient un taux de succès de 30,4%, déjà proche du taux de succès final. Au contraire, notre méthode se concentre sur les bords car la généralisation sur peu de données ne fonctionne pas bien, et obtient un score très faible de 1,1%. Au fur et à mesure que l'apprentissage avance, les deux méthodes se concentrent sur des zones similaires (notre méthode avec plus de variance) et obtiennent des taux de succès similaires : 35,5% pour notre méthode et 37,8% pour PPO.

Le fait que PPO non entraîné réussisse aussi bien nous a poussés à arrêter la comparaison (ce pourquoi il n'y a qu'une exécution de PPO dans la figure 4.4) pour nous interroger sur l'origine de ce succès. L'espace des solutions est centré sur le point du mur le plus proche de la base du robot, or ce point de contact est une bonne première solution. Le fait que PPO y arrive si bien est donc un artefact de ce choix et des méthodes d'exploration de PPO.

Nous avons donc testé en décalant latéralement l'espace de 2 m. PPO continue d'échantillonner autour du centre (qui est donc arbitrairement loin des bonnes solutions) et ne perçoit que des récompenses nulles. Du fait de sa mauvaise exploration, il n'arrive jamais à apprendre à résoudre la tâche avec une récompense éparse.

Une récompense continue Au lieu d'utiliser uniquement un booléen indiquant le succès ou l'échec au bout de 10 s, nous avons essayé avec une fonction continue qui correspond au temps que met le robot avant de chuter (avec donc un temps maximal de 10 s). C'est cette fonction qui est utilisée dans la version de MTMB-ME présentée dans la section 4.2. Cette fonction permet d'avoir un gradient de progression en apprenant des réflexes qui retardent de plus en plus la chute. Avec cette récompense, PPO arrive à converger vers de bonnes solutions même quand l'espace des solutions n'est pas centré sur de bonnes solutions.

Nous avons donc comparé le taux de succès des réflexes avec la main droite de notre méthode, qui apprend un classifieur à partir d'un jeu de données créé par MTMB-ME, PPO et, pour évaluer la difficulté de la tâche, une méthode de référence qui sélectionne aléatoirement le réflexe. La figure 4.5 montre une comparaison du taux de succès de la généralisation des trois méthodes au cours de l'apprentissage avec 10 réplications sur un jeu de données d'évaluations de 1000 situations échantillonnées aléatoirement. MTMB-ME est nettement meilleure avec peu de données et reste meilleure tout au long (p-value < 0.001) avec un taux de succès final de $39,0\%$ [premier quartile : $38,5$; troisième quartile : $39,5$], même si PPO se rapproche avec l'augmentation du nombre d'évaluations (taux de succès final : $34,6\%$ $[33,0; 36,0]$). Le taux de succès est "faible" (c'est-à-dire inférieur à 40%) car pour cette évaluation nous n'avons pas effectué de recherche exhaustive pour filtrer les situations pour lesquelles la chute n'est pas évitable (par exemple si le mur est trop loin ou le robot ne chute pas vers le mur).

4.3.2 Généralisation avec des environnements plus réalistes

Notre méthode n'a été évaluée que sur des surfaces planes et idéales. Bien qu'une grande partie des infrastructures humaines soit dotée de murs plats, un déploiement du réflexe dans le monde réel nécessite aussi l'applicabilité à des surfaces plus variées. Pour ce faire, dans cette sous-section, nous étudions l'applicabilité de la méthode à un environnement plus réaliste provenant de la modélisation 3D d'un environnement industriel réel.

4.3.2.1 Environnements plus réalistes en simulation

Comme environnement plus réaliste, nous avons pris une modélisation 3D sous forme de maillage provenant du jeu de données Gibson (Xia et al., 2018). Nous avons choisi l'environnement Webster (Figure 4.6) qui correspond à la reconstruction 3D d'une usine représentant un environnement industriel typique pour le déploiement de robots humanoïdes, par exemple lors de désastres industriels.

La figure 4.7 montre la distribution des surfaces de l'environnement Webster. Il contient principalement des surfaces verticales (des murs) et horizontales (des sols et plafonds) mais aussi de nombreuses surfaces "presque" verticales. C'est donc un environnement plus difficile par rapport à l'utilisation uniquement de surfaces planes verticales parfaites.

La première étape consiste à échantillonner dans ce maillage, contenant près de 500 000 triangles, des positions de contacts potentiels. Dans le cadre d'un déploiement sur le robot réel, il est peu probable que celui-ci ait accès à une reconstruction sous forme de maillage de son environnement. Le plus courant est d'utiliser une caméra de capture de profondeur (Z. Zhang, 2012) qui permet d'obtenir un nuage de points de l'environnement. Nous avons testé l'utilisation de l'algorithme de RANSAC (Fischler et al., 1981) avec ce nuage de points à partir du robot et avons trouvé des résultats encourageants mais qui ne seront pas davantage abordés. Dans le cadre de cette étude préliminaire, nous voulions principalement évaluer les capacités de généralisation dans un environnement plus complexe et non pour un déploiement sur un robot réel. Nous avons donc décidé de profiter d'avoir directement accès à ce maillage pour nous en servir pour déterminer des positions de contacts candidates. Le déploiement sur un robot réaliste devra faire l'objet d'une étude complémentaire.

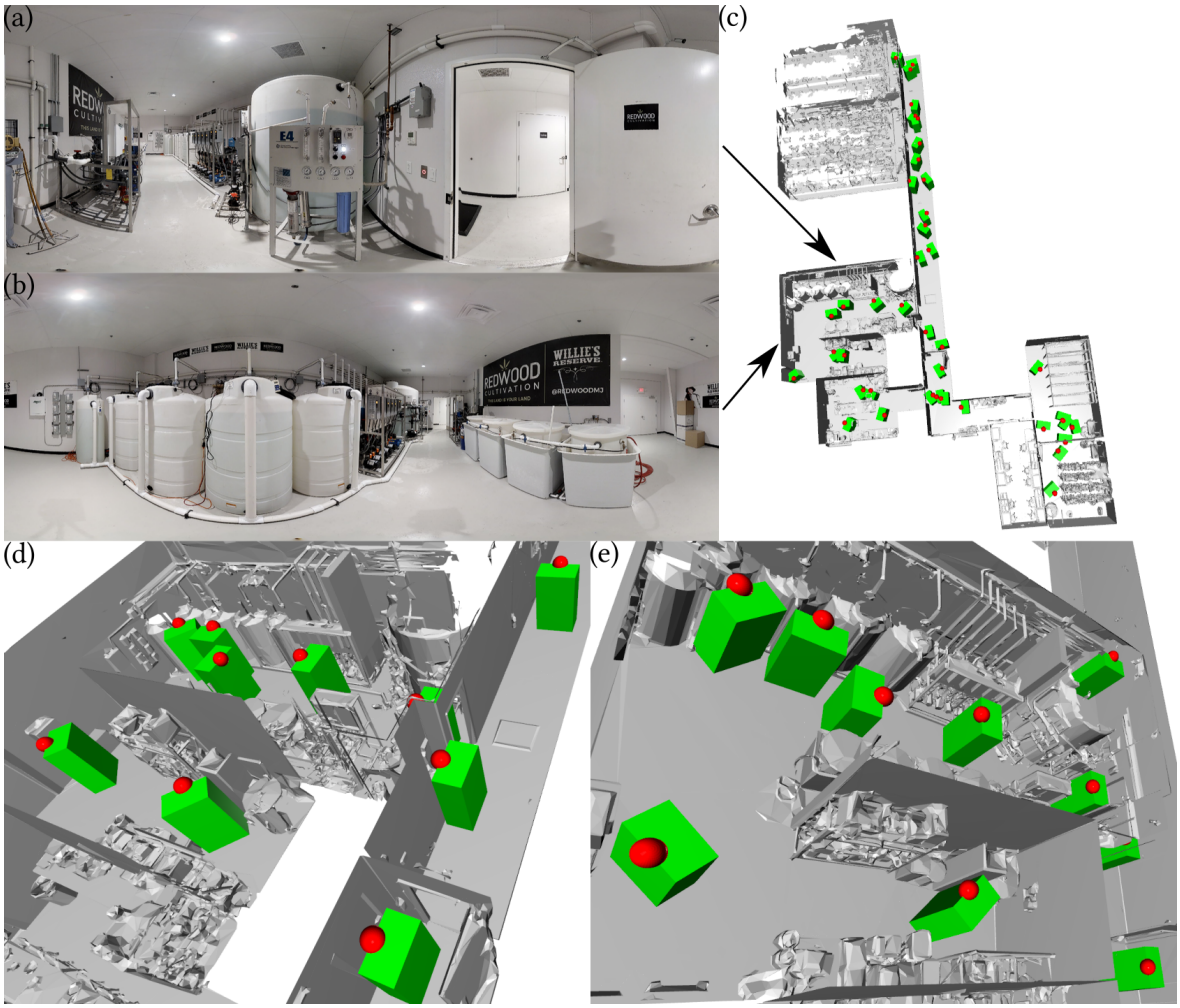

FIGURE 4.6 – (a-b) Photos de l'environnement Webster qui est scanné et reconstruit en modèle 3D. (c) Vue globale du maillage de l'environnement Webster. (d-e) Zoom dans une des salles avec des positions possibles du Talos. Les pavés verts représentent une forme englobante autour du robot et la boule rouge sa tête. Les positions sont échantillonnées aléatoirement puis filtrées par rejet de façon à ce que le robot ne soit pas en pénétration avec l'environnement et qu'il y ait une surface proche de son bras droit.

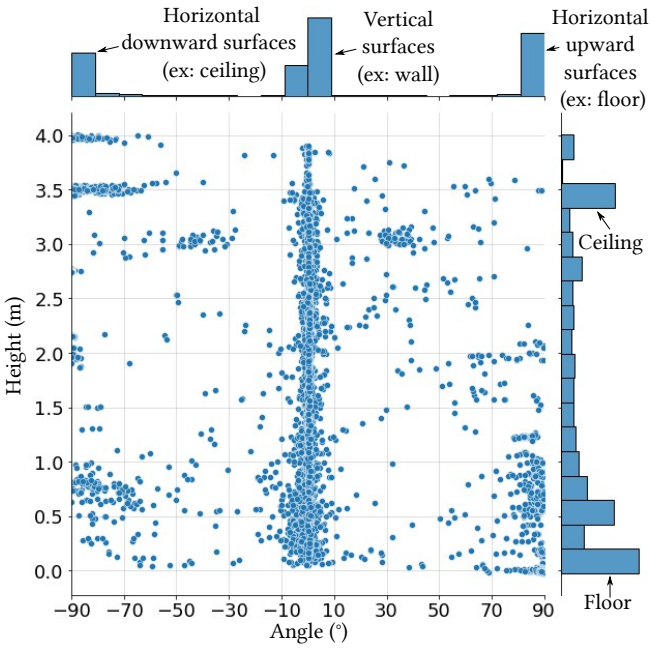

FIGURE 4.7 – Distribution des surfaces se trouvant dans l'environnement Webster. Il y a principalement des regroupements de surfaces horizontales au niveau des plafonds et sols, et des surfaces verticales (par exemple les murs) à toutes hauteurs. Il y a aussi des surfaces plus irrégulières provenant de l'infrastructure pour lesquelles une méthode généraliste doit pouvoir s'appliquer si elles sont la seule source de support possible.

Sélection des positions et orientations valides. Pour sélectionner une position et une orientation auxquelles placer le Talos à partir du maillage, nous utilisons l'algorithme suivant :

1. déterminer les dimensions du sol avec des rectangles englobants $R_{1:n}$;
2. déterminer une boîte englobante du robot B_{robot} (hauteur : 1,7 m, largeur : 1,0 m, profondeur : 0,6 m) ;
3. déterminer une boîte englobante de portée de bras du robot B_{reach} (à moins de 1,2 m de l'épaule et entre 0,5 m et 2 m de hauteur) ;
4. échantillonner aléatoirement une position en 2D (x, y) sur le sol et une orientation r_z autour de l'axe vertical ;
5. filtrer pour que (x, y) appartienne bien à l'un des rectangles englobants du sol R_i ("Est-ce que la position se trouve bien sur du sol ?") ;
6. filtrer pour qu'il n'existe pas de point du maillage à l'intérieur de B_{robot} ("Est-ce que le robot n'est pas en collision avec l'environnement ?") ;
7. filtrer pour qu'il existe bien des points du maillage à l'intérieur de B_{reach} ("Est-ce que le robot peut atteindre une surface de support depuis cette position ?").

Déterminer les potentielles surfaces de contacts. Comme précédemment, l'algorithme s'entraîne avec des surfaces planes parfaites (qui sont donc générées via le code) dans le but de pouvoir généraliser à des surfaces quelconques. Ainsi, pour l'étape d'évaluation de la généralisation, l'algorithme pré-sélectionne des positions de contacts sur des surfaces "suffisamment planes" dirigées vers le robot. Ces positions sont ensuite envoyées au classifieur qui permet de sélectionner la position de contact la plus prometteuse pour permettre de stabiliser le robot.

Pour déterminer des positions de contacts candidates, nous exécutons l'algorithme suivant :

1. création des classes d'équivalence des triangles du maillage en regroupant dans la même classe les triangles adjacents les uns aux autres et ayant un vecteur normal semblable, c'est-à-dire de produit scalaire inférieur à $\varepsilon = 0,01$ ("ne garder que les surfaces suffisamment planes") ;

2. calcule pour chaque classe de la normale moyenne et la surface englobante puis filtrer des surfaces trop petites, (c'est-à-dire n'incluant pas un cercle de rayon 20 cm et de surface inférieure à 0,04 cm^2), et des surfaces non verticales, (c'est-à-dire dont la composante verticale de la norme est supérieure à 0,05) ("ne garder que les surfaces suffisamment grandes et verticales") ;
3. échantillonnage pour chaque classe restante et pour chaque triangle de cette classe des points appartenant à ce triangle avec une densité d'échantillonnage de 1 point par cm^2.

Comme les triangles d'une même classe n'ont pas tous la même normale, le processus de création des classes d'équivalences est déterministe uniquement pour un ordre de parcours des triangles.

4.3.2.2 Le problème de la robustesse des solutions

Un premier problème qui apparaît lorsque nous passons d'une surface plane parfaite à une surface quelconque est la robustesse du contact. En effet, avec un plan supposé infini par rapport à la portée du robot, la position réelle atteinte par la main importe peu tant que le contact rend le robot plus stable.

La figure 4.8.a montre, pour une même situation (c'est-à-dire position et orientation du mur autour du robot et sa posture), les points de contacts atteints sur le mur pour une même commande exécutée pour l'ensemble des dommages. D'une part, pour une même commande, les contacts atteints sont variés, montrant qu'une unique commande implique un unique point de contact. D'autre part, un même réflexe réussit ou échoue en fonction du dommage. Il n'existe donc pas de réflexe unique pour l'ensemble des dommages. De plus, la différence de position du contact atteint ne permet pas d'expliquer le fait qu'un réflexe va fonctionner, car il y a des contacts réussis et échoués aux mêmes positions. Ceci s'explique par la différence de dynamique : en effet, même si le robot est contrôlé en position avec une hypothèse de dynamique quasi-statique, le dommage entraîne une dynamique de chute non quasi-statique.

La figure 4.8.b montre de façon similaire pour 16 situations différentes les points de contacts atteints pour différentes commandes, en indiquant la différence entre la commande et le point de contact atteint. Il peut y avoir une grande différence entre la commande correspondant à une position de contact cible sur le mur et la position de contact réellement atteinte.

Cette différence pose problème car, pour pouvoir généraliser à un environnement plus complexe où le mur n'est pas supposé infini, il faut pouvoir s'assurer que le robot atteindra un point de contact qui existe dans l'environnement. Par exemple, si nous choisissons comme commande un contact sur une poutre, selon le dommage, il est possible que le robot ne parvienne pas à entrer en contact avec la poutre.

4.3.2.3 Méthode de résolution : double classifieurs

Est-il possible de construire l'archive de solutions avec des murs parfaits ? Une des premières questions que nous nous sommes posées est de savoir si il est pertinent de construire l'archive de solutions directement sur l'environnement réaliste. En effet, comme nous voulons garder la même représentation de l'environnement pour le classifieur (c'est-à-dire une surface plane décrite par sa position et son orientation par rapport au robot) même avec l'étape de filtrage, deux maillages légèrement différents peuvent avoir la même représentation.

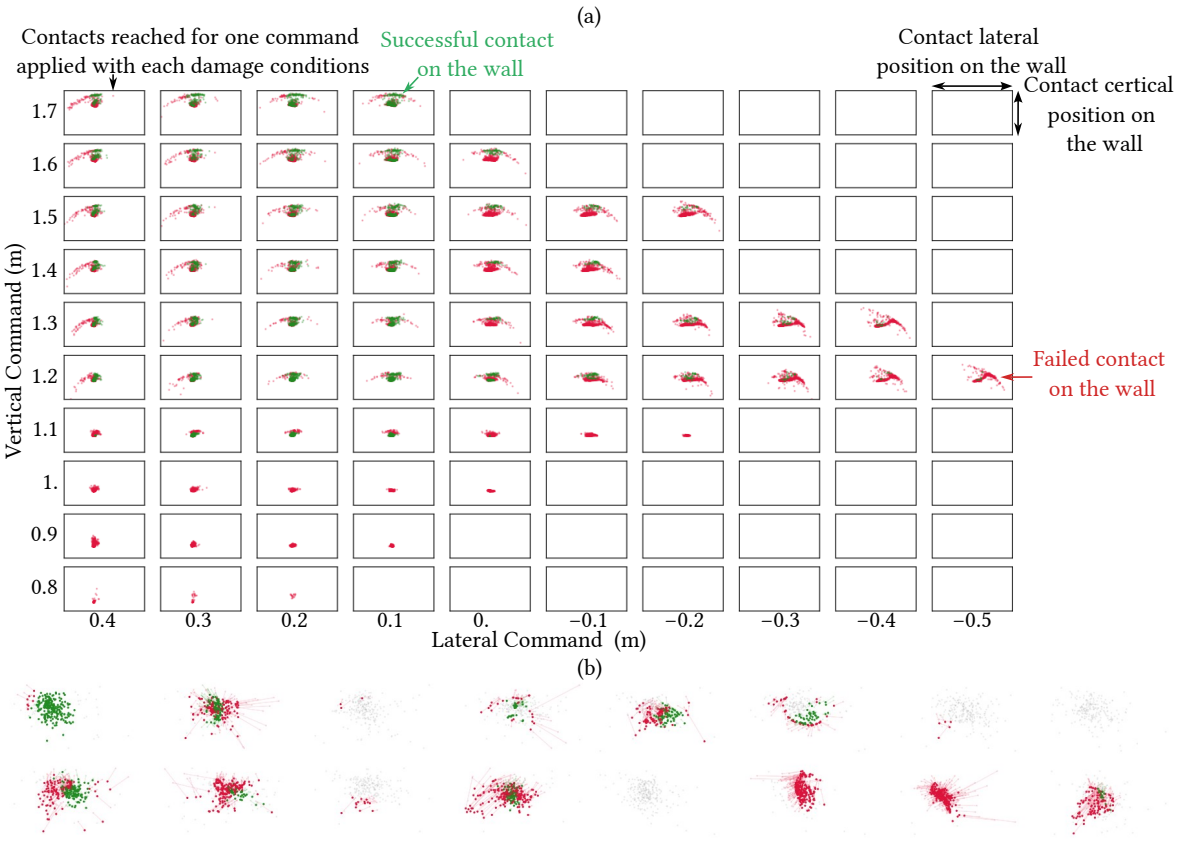

FIGURE 4.8 – (a) Nuage de points des positions de contact sur le mur atteints étant donné une même situation, différentes commandes (une commande = une boîte) et pour chaque dommage possible (les points dans chaque boîte). Point vert = le contact a permis de stabiliser le robot, c'est-à-dire le réflexe est réussi. Point rouge = le contact n'a pas stabilisé le robot, c'est-à-dire le réflexe est raté. Pas de point = pas de contact avec le mur, c'est-à-dire le réflexe est raté. (b) Nuage de points de la position de contact sur le mur atteints pour différentes commandes dans différentes situations (chaque nuage correspond à plusieurs commandes pour la même situation). Les points gris correspondent aux commandes, les points verts aux contacts avec le mur ayant réussis et les points rouges aux contacts ayant échoués. Les flèches lient les commandes aux contacts atteints et montrent que le contact atteint peut se situer à plusieurs dizaines de centimètres.

Comme la dynamique de contact de notre simulateur est assez chaotique et que MTME-ME est un algorithme élitiste, il est possible qu'il trouve une solution qui fonctionne "par chance" pour une maillage mais qui ne se généralise pas à d'autres maillages.

Une première étude que nous avons faite est de regarder si les solutions pour un mur parfait correspondent aux solutions pour une surface "plane" du maillage. La figure 4.9.a illustre pour une situation donnée la différence entre l'ensemble des contacts obtenus sur le maillage et sur un plan parfait pour les mêmes commandes. Les contacts réussis à l'intérieur de la surface se trouvent aux mêmes endroits sur le maillage et sur le plan parfait.

Pour comparer plus quantitativement, nous définissons la robustesse d'un réflexe comme le taux de dommages qu'il permet d'éviter. La figure 4.9.b compare la robustesse des réflexes pour des surfaces "planes" de l'environnement réaliste et les plans parfaits correspondants. Il y a une bonne corrélation, ce qui légitime, selon nous, l'emploi d'un entraînement avec des murs parfaits pour généraliser ensuite sur l'environnement réaliste.

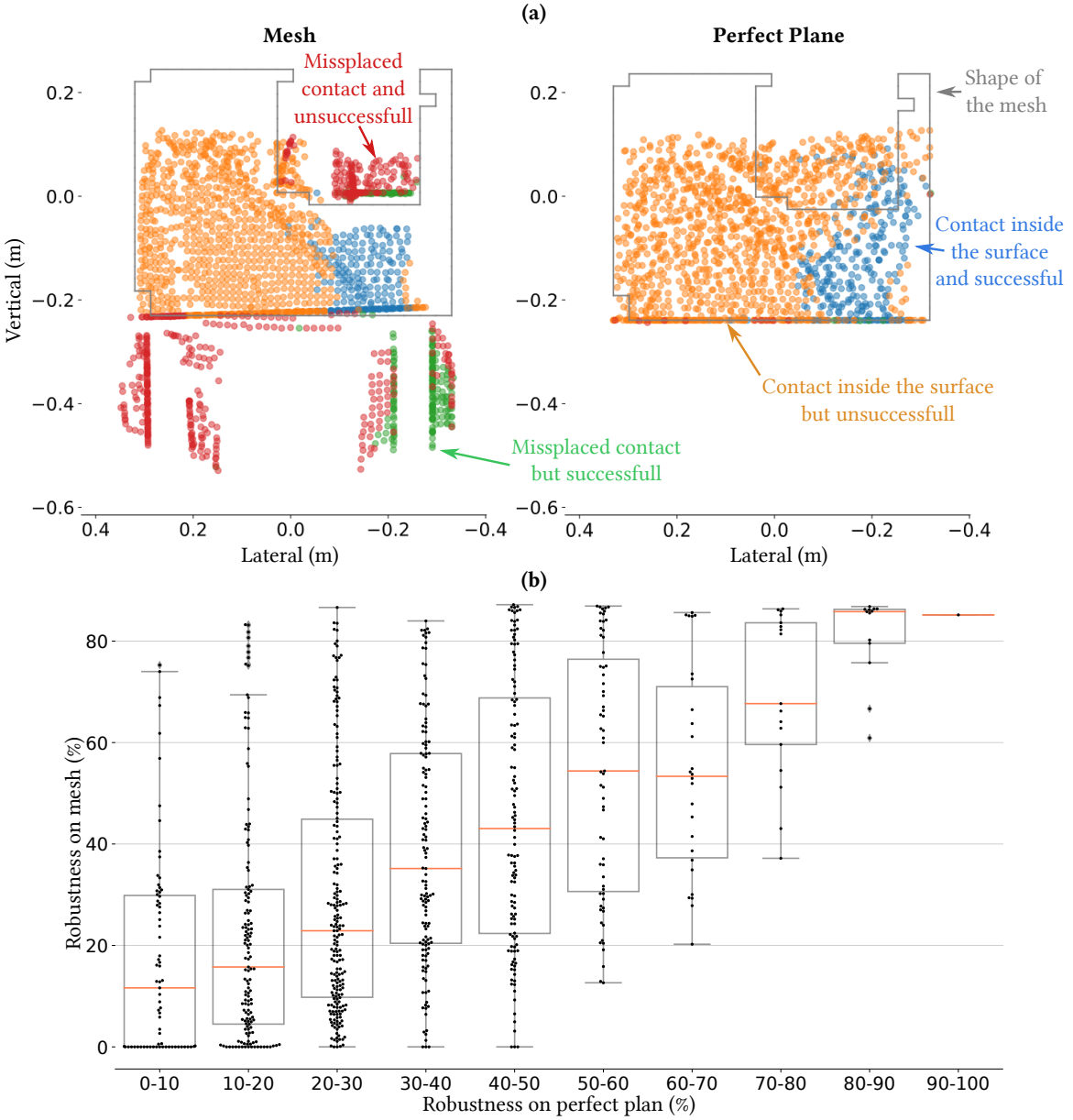

FIGURE 4.9 – (a) Comparaison des points de contacts entre une surface "plane" dans l'environnement réaliste et un mur parfait à la même position et orientation. Nous distinguons : les contacts sur la surface en question et hors de la surface (dans l'environnement réaliste, le bras du robot peut entrer en contact avec l'environnement), et les contacts qui permettent aux robots de retrouver l'équilibre et ceux qui ne le permettent pas. (b) Comparaison entre la robustesse des solutions sur des surfaces "planes" de l'environnement réaliste et les plans parfaits équivalents.

Est-il possible d'améliorer la robustesse des solutions pendant la généralisation ? L'environnement réaliste pose un autre problème : comme les surfaces ne sont plus supposées infinies, il faut que le réflexe du robot lui fasse atteindre la surface (si, par exemple, il y a un trou ou un obstacle dans la surface, il faut que le robot puisse l'éviter). Il faut donc pouvoir choisir plus précisément la position de contact atteinte par le robot. De plus, une hypothèse était que le succès d'un réflexe est plus corrélé à la position de contact atteinte sur la surface qu'à la commande envoyée.

Une idée que nous avons testée mais qui n'a pas été concluante a été de séparer le travail du classifieur en deux étapes. Dans le chapitre 3, le classifieur prédit pour une commande si elle va être un succès ou non (**C2S** pour *Command to Success*). L'alternative que nous avons testée est (1) d'avoir un classifieur (**C2P** pour *Command to Position*) qui prédit pour une commande une carte de probabilité d'atteindre chaque position de contact et (2) un classifieur (**P2S** pour *Position to Success*) qui prédit pour une position de contact atteinte si elle va être un succès ou non.

Mettre bout à bout ces deux classifieurs permet de revenir à une prédiction pour une commande du taux de succès. De plus, cela permet, pendant l'inférence pour une surface particulière, de pénaliser les commandes qui ont une forte chance d'aboutir à un contact hors de cette surface. Autrement dit, plutôt que d'avoir appris pour "toutes" les surfaces, les classifieurs peuvent apprendre avec des surfaces "parfaites" et s'adapter lors de l'inférence.

La figure 4.10 montre la chute de performance des différentes méthodes à sélectionner une commande robuste lorsque les positions de contact atteintes sont rejetées si elles sont trop éloignées de la commande. C'est-à-dire que les positions de contacts atteintes doivent respecter un minimum de précision. La performance du classifieur direct ClassifieurC2S chute rapidement, ce qui légitime de trouver une autre méthode. Notre méthode en deux classifieurs (ClassifieurC2P et ClassifierP2S) a une plus faible chute de performance. Surtout, la méthode avec les données réelles (OracleC2P et $\text{Oracle}_{\text{P2S}}$, qui correspond à un cas où les classifieurs sont supposés parfaits) obtient une performance très bonne (c'est-à-dire supérieure à 90%) et est quasi constante, montrant qu'elle permet théoriquement de choisir avec précision la commande la plus robuste.

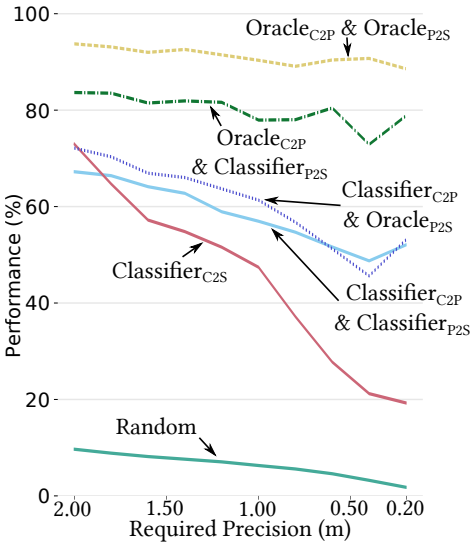

FIGURE 4.10 – Comparaison des performances des différentes méthodes lorsque les positions de contacts atteintes sont rejetées si elles sont arbitrairement loin de la commande (Précision).

Sans entrer dans les détails étant donné l'échec de la méthode, la figure 4.11.a montre sa performance pour sélectionner la commande la plus robuste. La méthode avec les données réelles (OracleC2P et OracleP2S) obtient une médiane de performance supérieure à 90%, ce qui montre sa validité théorique. Un classifieur direct ClassifieurC2S obtient une assez bonne performance, supérieure à 80% de médiane. La nouvelle méthode avec les deux classifieurs (ClassifieurC2P et ClassifierP2S) obtient une performance médiane similaire à la méthode de référence aléatoire. La figure 4.11.b montre des exemples de cartes de succès de différentes commandes pour trois situations. Les deux classifieurs introduits (et particulièrement ClassifieurC2P) diminuent la qualité de la carte.

Comme la méthode OracleC2P et ClassifierP2S est assez proche du classifieur direct, nous avons conclu que c'est le classifieur **C2P** qui pose problème. Cependant, comme le classifieur direct est assez bon, nous avons décidé de revenir à ce mode d'inférence pour choisir quelle commande effectuer.

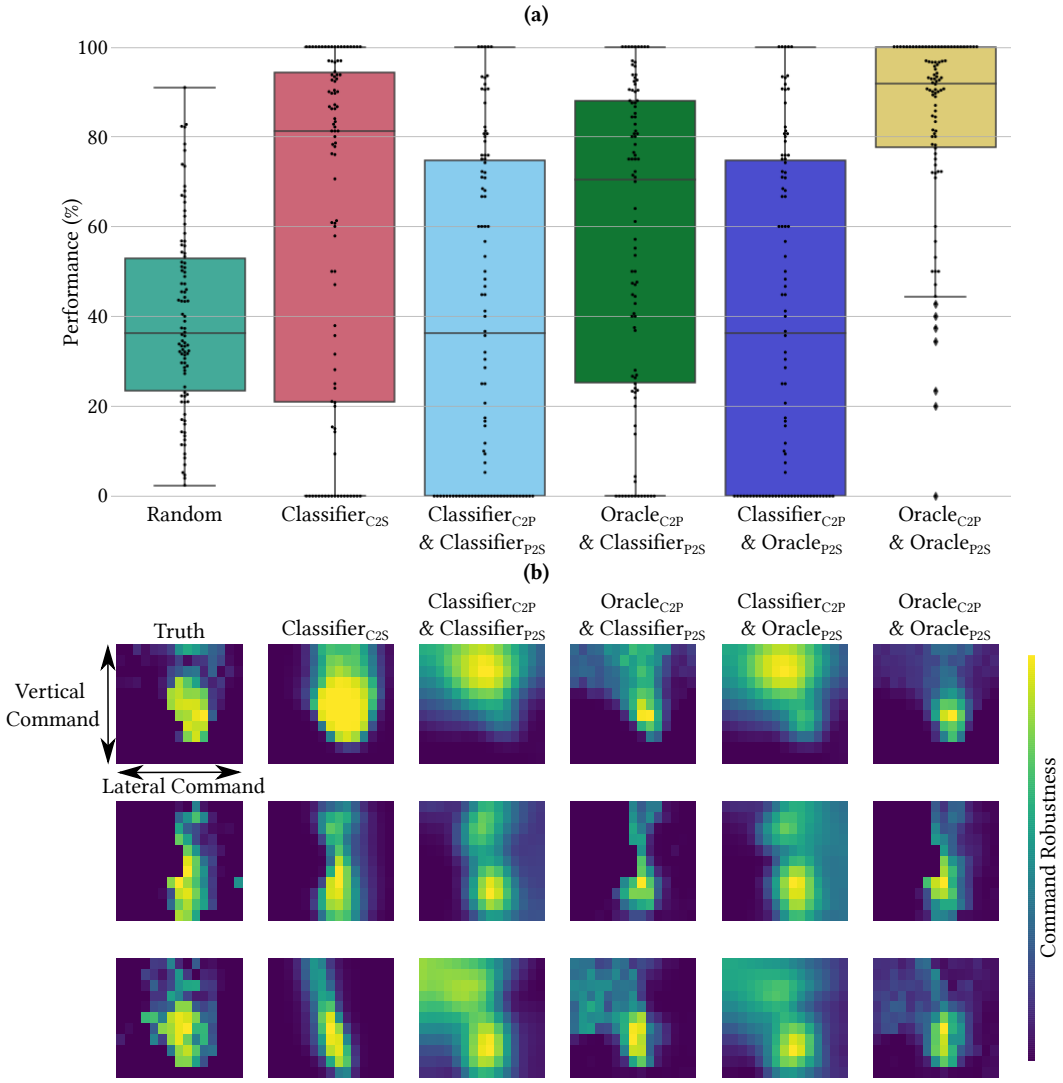

FIGURE 4.11 – (a) Comparaison de la performance à sélectionner une commande robuste (normalisée par rapport à la commande la plus robuste de chaque situation). Nous comparons une méthode de référence qui choisit aléatoirement, un classifieur direct **C2S** comme dans le chapitre 3, un classifieur en deux étapes **C2P** puis **P2S**, et des oracles remplaçant une ou les deux parties avec les données réelles. (b) Exemple de carte de prédiction des commandes en comparaison avec la vérité. L'échelle des couleurs correspond au min et max de chaque carte donc ne permet pas la comparaison des valeurs mais permet de comparer le point le plus probable pour chacune.

4.3.2.4 Résultats

Ayant conclu qu'utiliser un seul classifieur, comme dans le chapitre 3, permet d'obtenir une robustesse satisfaisante, nous avons décidé de comparer la généralisation de cette méthode.

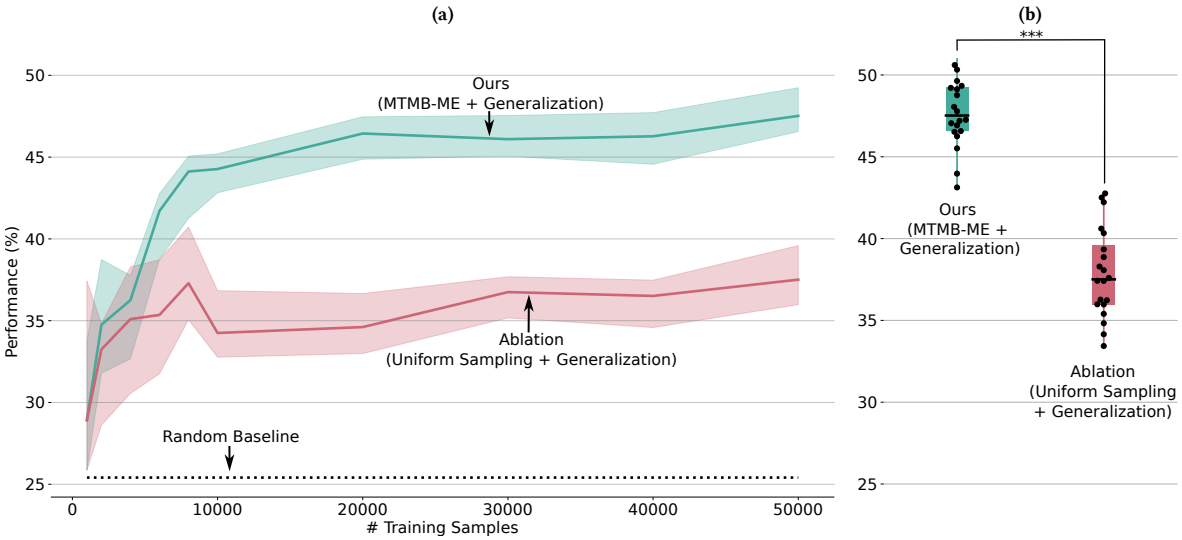

FIGURE 4.12 – Évaluation de la généralisation pour sélectionner un réflexe robuste dans un environnement plus réaliste de notre méthode (MTMB-ME + classifieur), d'une ablation sans MTMB-ME (exploration aléatoire + classifieur) et d'une méthode de référence aléatoire. (a) La performance en fonction de la taille du jeu de données pour entraîner le classifieur. Le trait plein représente la médiane et la zone colorée le premier et troisième quartile de 20 réplications. (b) Boîte à moustaches de l'évaluation avec le plus de données (c'est-à-dire 50 000 évaluations).

Méthodes de référence. Pour choisir un réflexe, notre méthode échantillonne et filtre des points de contacts candidats puis utilise le classifieur pour sélectionner celui qui a le plus de chances de réussir à équilibrer le robot. Il ne paraît pas trivial de faire la même chose avec PPO. En particulier, il est difficile de savoir quelle représentation choisir pour l'environnement sous forme de maillage ou pour choisir parmi une liste de candidats de taille variable. Selon nous, ces questions forment un travail de recherche en elles-mêmes qui n'est pas l'objet de cette thèse. Nous avons donc décidé de ne pas comparer notre méthode à PPO.

À la place, nous comparons notre méthode (c'est-à-dire la création d'un jeu de données avec MTMB-ME puis généralisation avec un classifieur) avec la création d'un jeu de données en utilisant un échantillonnage uniforme des commandes puis généralisation avec un classifieur. Cette ablation de MTMB-ME permet d'évaluer si effectivement MTMB-ME permet d'obtenir un meilleur jeu de données pour la généralisation.

Chaque classifieur est entraîné avec la même méthodologie qu'au chapitre 3 : 100 époques avec la fonction de coût *WeightedBinaryCrossEntropy* et une validation utilisant le coefficient de corrélation de Matthews (Matthews, 1975). Nous utilisons une fonction de coût pondérée car elle permet de corriger le déséquilibre entre les exemples positifs (succès du réflexe) bien moins nombreux que les exemples négatifs (échec du réflexe).

En plus de cela, pour évaluer la difficulté du problème et montrer que notre méthode aide à le résoudre, nous comparons à une méthode de référence qui sélectionne aléatoirement le réflexe parmi les positions candidates au lieu d'utiliser un classifieur.

Évaluation. Pour évaluer les différentes méthodes, l'algorithme échantillonne d'abord 230 situations composées d'un emplacement dans l'environnement complexe et d'une posture, position, et orientation du robot à cet emplacement. Il échantillonne ensuite à cet emplacement, étant donnée l'orientation du robot, un ensemble de positions de contacts possibles sur le maillage (c'est-à-dire appartenant à des surfaces suffisamment planes, verticales et orientées vers le robot) avec une densité moyenne de 100 contacts par m^2. Cela correspond en moyenne

à 100 contacts candidats par situation et 23 142 réflexes au total. Et pour chacune de ces commandes, nous testons avec 16 dommages différents pour évaluer la robustesse du réflexe. Ceci fait un total de 370 272 évaluations de réflexe en simulation.

Comme notre classifieur sélectionne parmi un ensemble de réflexes candidats, nous pouvons nous permettre d'évaluer chaque candidat indépendamment de la méthode qui le choisi. Ceci permet de créer le jeu de données d'évaluation en une fois, au lieu de devoir évaluer chaque réflexe en simulation. En effet, une fois que la méthode choisi un réflexe, nous pouvons directement obtenir sa robustesse sans avoir à réévaluer le réflexe.

Après avoir enlevé les situations qui n'ont aucune solution, il reste 182 situations avec en moyenne 112 candidats, soit 20 430 candidats au total. La mesure de performance correspond à normaliser la robustesse du réflexe choisi par rapport au réflexe de robustesse optimal. C'est-à-dire qu'une performance de 100% indique que la méthode a choisi le meilleur candidat à chaque fois.

La figure 4.12 montre la comparaison entre les trois méthodes. Notre méthode (47,5% [46,6 ; 49,2]) surpasse significativement l'ablation (37,5% [36,0 ; 39,6]) et la méthode de référence (p-value $< 0{,}05$ avec un test de Mann-Whitney U).

4.4 Conclusion

Nous avons présenté une nouvelle méthode de qualité-diversité multi-tâche, MTMB-ME, qui permet d'obtenir : (1) un jeu de données de solutions plus dense et plus varié que les méthodes naïves, (2) une meilleure performance par rapport à PPO sur des murs parfaits avec un seul bras et (3) une meilleure généralisation à un environnement réaliste par rapport à un jeu de données créé par une recherche aléatoire.

Un axe de recherche en robotique qui pourrait permettre d'étendre notre méthode est l'amélioration du réflexe. En effet, il est principalement utile pour des dommages à une jambe avec la présence d'un mur du même côté. Une première amélioration pourrait être d'utiliser du contrôle de corps complet multi-contact (Rouxel et al., 2023) pour permettre de mieux répartir l'effort sur les deux mains et éviter des problèmes de rebond et de glissement. Une seconde amélioration pourrait être d'utiliser une autre forme de réflexe que le contact, comme un réflexe de saisie, par exemple en agrippant un rebord comme le ferait un humain.

De plus, dans le cas d'un déploiement dans un environnement complexe, le robot aurait besoin de construire une carte 3D de son environnement, par exemple à l'aide d'un algorithme de *SLAM* et de reconstruction 3D. Une question serait d'étudier comment chercher des éléments de l'environnement comme points de support pour éviter la chute. Par exemple, à partir d'une carte 3D générée avec l'aide de l'un de ces algorithmes donnée en entrée à un réseau convolutif. Une telle architecture pourrait permettre la comparaison avec un algorithme d'apprentissage par renforcement profond qui prendrait ces mêmes données pour générer directement un réflexe.

Du point de vue de la première étape de résolution, une première limitation de MTMB-ME est qu'elle ne permet de résoudre qu'un nombre fini de tâches prédéfinies. De plus, augmenter le nombre de tâches à résoudre augmente le nombre de cellules dans l'archive, ce qui réduit la vitesse de convergence. Une seconde limitation est qu'elle nécessite la définition d'un espace de comportements. La diversité de comportement permet, au moment de l'inférence, d'avoir rencontré suffisamment de solutions pour pouvoir s'adapter, par exemple, à une géométrie particulière de l'environnement. L'espace de comportements n'est pas toujours facile à définir

pour tous les problèmes. Dans le chapitre suivant, nous étudierons la conception d'un nouvel algorithme multi-tâche qui essaie de pallier ces différentes limitations, en particulier pour pouvoir s'appliquer à n'importe quel problème d'optimisation multi-tâche.

Chapitre 5

Passer d'une étape de résolution discrète à une résolution continue

Contribution : Ce chapitre s'appui sur le papier *Parametric-Task MAP-Elites* (Anne et Mouret, 2024) sur lequel j'ai travaillé sous la supervision de Jean-Baptiste Mouret. Il ajoute une comparaison de plusieurs méthodes de généralisation du jeu de données (Sections 5.4.5 et 5.5.4.2).

Sommaire

5.1 Introduction

Jusqu'à maintenant, nous nous sommes intéressés à apprendre des comportements en découpant l'apprentissage en une phase de résolution d'un ensemble de tâches suivie d'une généralisation des solutions. Pour résoudre l'ensemble de tâches, nous avons utilisé dans le chapitre 3 une optimisation par recherche en grille sur chaque tâche séparément et, dans le chapitre 4, un algorithme de qualité-diversité multi-tâche.

Dans ce chapitre, nous abandonnons le problème de qualité-diversité présenté dans le chapitre précédent pour nous concentrer sur l'optimisation (Fletcher, 1987 ; Chong et al., 2013) qui est plus générale et omniprésente dans nos sociétés. Par exemple : dans le transport, pour trouver le chemin le plus court, qui consomme le moins de carburant ou qui coûte le moins cher (Hart et al., 1968 ; Herty et al., 2003) ; en construction, pour satisfaire les normes de sécurité tout en minimisant les coûts (Piryonesi et al., 2017) ; en conception, pour concevoir des véhicules les plus aérodynamiques ; en mécanique, pour contrôler des robots formés de solides rigides articulés (Vereshchagin, 1989) ; en économie, pour étudier la dynamique entre les consommateurs qui maximisent leur utilité et les entreprises qui maximisent leur profit à l'aide de la théorie du contrôle (Dorfman, 1969). Dans l'industrie, pour contrôler les processus de fabrications (Kumar et al., 2012) ou de raffinage (J. Z. Lu, 2015) à l'aide de la commande prédictive.

Nous pouvons distinguer trois familles de méthodes : les méthodes directes (ou exactes) qui renvoient la valeur exacte de l'extremum (Fletcher, 1987 ; Chong et al., 2013), les méthodes itératives avec preuves de convergence (Fletcher, 1987 ; Chong et al., 2013), et les heuristiques (Pearl, 1984). Les méthodes directes et les méthodes itératives avec preuve de convergence nécessitent des hypothèses plus fortes, en particulier que la fonction f soit linéaire, quadratique, convexe, ou au moins différentiable. Bien que cela soit le cas dans de nombreux domaines, de nombreux problèmes nécessitent l'optimisation d'une fonction non-linéaire, non-quadratique, non-convexe, non-différentiable, dont la seule information accessible est d'évaluer la fonction pour obtenir sa valeur en un point.

Ces fonctions dites boîtes noires sont le plus souvent rencontrées lorsque l'évaluation de la fonction nécessite un système réel, par exemple un humain pour optimiser une recette de cookies (Solnik et al., 2017) ou un système complexe comme une simulation, par exemple une simulation de procédés chimiques (Audet et al., 2008). Elles ont aussi des applications dans de nombreux domaines (Alarie, Audet, A. E. Gheribi et al., 2021) tels que la transmission et distribution d'énergie (Alarie, Audet, Garnier et al., 2013), la conception de nouveaux matériaux (A. Gheribi et al., 2018), l'assemblage de pièces d'avion (Lupuleac et al., 2020), les géométries cardiovasculaires (Marsden et al., 2008) ou encore en astrophysique (Aasi et al., 2013). Les méthodes d'optimisation boîte noire ont été développées en partie pour répondre au besoin d'optimisation des problèmes complexes où la fonction f est accessible uniquement par évaluation. Ce n'est pas leur seul intérêt, n'ayant pas accès aux gradients pour guider la recherche d'extremums, ces méthodes ont développé des techniques d'exploration qui leur permettent, par exemple, d'être plus robustes aux extremums locaux.

Les méthodes heuristiques s'appliquent aux problèmes d'optimisation boîtes noires. Le calcul évolutionniste (Back et al., 1997) propose de nombreuses heuristiques, telles que les stratégies d'évolution (Section 2.4). Une méthode considérée comme l'état de l'art est CMA-ES (Hansen et al., 2003) (Section 2.4.1.2).

Il existe de nombreuses méthodes résolvant le problème d'optimisation pour une fonction f. Mais que faire lorsque nous avons plusieurs fonctions à optimiser ? En effet, bien que de nombreux problèmes ne nécessitent que d'optimiser une seule fonction de coût, de nombreux autres nécessitent d'optimiser un ensemble de fonctions, chacune correspondant à une "tâche" différente. Par exemple en robotique, il peut être utile d'apprendre à marcher pour différentes morphologies de robots (Mouret et Maguire, 2020), ce qui signifie que chaque tâche est un problème d'optimisation avec une simulation de robot différent. Au lieu de résoudre chaque optimisation indépendamment, exploiter les similarités des solutions en résolvant toutes les tâches simultanément permet de diminuer le temps de calcul requis pour résoudre l'ensemble des tâches. En apprentissage, il peut aussi être intéressant d'ajuster finement les hyper-paramètres d'un algorithme pour différents jeux de données. Dans ce cas, chaque tâche correspond à minimiser la fonction de coût pour un jeu de données (S. Huang et al., 2021).

Les problèmes multi-tâches sont présents dans de nombreux autres domaines tels que la robotique évolutionniste (Mouret et Maguire, 2020 ; C. Wang et al., 2022 ; Bali, Y.-S. Ong et al., 2020) ; la science de la donnée pour de la régression symbolique de différents jeux de données réels en utilisant de la programmation génétique (Zhong, Feng et al., 2020) ; le test de plusieurs logiciels en utilisant de l'évolution multi-factorielle (Sagarna et al., 2016) ; la planification logistique pour différents problèmes de routage (Feng, Y. Huang et al., 2021) ; et la conception pour les voitures (Yokoya et al., 2019), l'emballage (Dai et al., 2022), les modèles photovoltaïques (J. Liang et al., 2020) ou la répartition du réseau électrique (J. Liu et al., 2020).

À notre connaissance, tous les algorithmes d'optimisation multi-tâches ne résolvent qu'un ensemble fini de tâches alors qu'il existe de nombreux problèmes continus. Les algorithmes d'optimisation multi-tâches doivent donc discrétiser ces espaces de tâches continues. Par exemple, en robotique, la morphologie d'un robot est caractérisée par des longueurs continues de membres et des limites articulaires continues. Le choix de discrétisation est donc forcément arbitraire. Lorsque la fonction à optimiser est différentiable, il existe quelques approches (c'est-à-dire l'optimisation paramétrique présentée dans la section 5.3.1) pour résoudre le problème d'optimisation multi-tâches continue, que nous appelons le problème d'optimisation de tâches paramétriques, mais nous ne sommes pas au courant d'un tel algorithme pour l'optimisation boîte noire.

La contribution de ce chapitre est un algorithme d'optimisation de tâches paramétriques boîte noire, que nous appelons *Parametric-Task MAP-Elites* (PT-ME). PT-ME s'inspire de *Multi-Task MAP-Elites* (MT-ME) (Mouret et Maguire, 2020) un algorithme multi-tâche élitiste. L'idée centrale est de concevoir un algorithme qui résout autant de tâche que possible avec le budget alloué, résolvant asymptotiquement le problème d'optimisation de tâches paramétriques en recouvrant l'espace des tâches de solutions de haute qualité. Le jeu de données résultant peut ensuite être distillé, par exemple avec un réseau de neurones profonds, permettant ainsi de généraliser et d'avoir accès à une solution de haute qualité pour n'importe quelle instance de tâche du problème.

Les principales caractéristiques de PT-ME sont :

— il échantillonne une nouvelle tâche à chaque itération, c'est-à-dire que plus son budget d'évaluation est important, plus il résoudra de tâches ;
— sa pression de sélection est décorrélée du nombre de tâches qu'il résout, c'est-à-dire que son efficacité ne décroît pas avec le nombre de tâches à résoudre ;
— il utilise un opérateur de variation spécialement conçu pour exploiter la structure multi-tâche du problème afin d'améliorer sa performance ;

— il distille les solutions en une approximation de fonction continue, c'est-à-dire un réseau de neurones, résolvant ainsi le problème d'optimisation de tâches paramétriques.

Nous évaluons notre méthode ainsi que ses ablations sur deux problèmes jouets et un problème plus réaliste d'optimisation de tâches paramétriques : *10-DoF-Arm* (utilisé dans Mouret et Maguire (2020), C. Wang et al. (2022) et Huynh Thi Thanh et al. (2023a)), *Archery* (un problème plus difficile, car ayant une récompense éparse) et *Door-Pulling* (consistant en un robot humanoïde ouvrant une porte en tirant sur une poignée verticale). Nous comparons notre méthode à plusieurs méthodes de référence et montrons que notre méthode résout le problème d'optimisation de tâches paramétriques en étant plus performante que toutes les méthodes de référence et ablations.

5.2 Formulation du problème

Le but est de trouver une fonction G qui renvoie la solution optimale pour chaque paramétrisation de tâche θ. Formellement :

$$\forall \theta \in \Theta, G(\theta) = x^*_\theta = \underset{x \in \mathcal{X}}{\operatorname{argmax}}(f(x, \theta)) \tag{5.1}$$

où $\mathcal{X}$ est l'espace des variables, Θ est l'espace de paramétrisation de tâche, $f : \mathcal{X} \times \Theta \to \mathbb{R}$ la fonction à optimiser (aussi appelée fonction de *fitness*), et $G : \Theta \to \mathcal{X}$ une fonction qui renvoie la variable d'optimisation x^*_θ qui maximise la fonction de *fitness* pour la paramétrisation de tâche θ. La différence principale avec l'optimisation multi-tâche est que l'espace représentant les tâches Θ est continu et non plus un ensemble fini de tâches.

Pour simplifier sans perte de généralité, dans le reste du chapitre, nous supposerons que l'espace des variables $\mathcal{X} = [0, 1]^{d_x}$ où d_x est sa dimension et que l'espace de paramétrisation de tâche $\Theta = [0, 1]^{d_\theta}$ où d_θ est sa dimension.

5.3 Travaux connexes

5.3.1 Optimisation paramétrique

Optimiser une fonction paramétrée par un ou plusieurs paramètres discrets ou continus est étudié en mathématiques et est appelée la programmation paramétrique (ou multi-paramétrique lorsqu'il y a plusieurs paramètres) (Barnett, 1968 ; Gal et al., 1972 ; Fiacco, 1976 ; Pistikopoulos et al., 2000 ; Pappas et al., 2021). La plupart des algorithmes divisent l'espace des paramètres en plusieurs régions critiques et résolvent le problème d'optimisation sur chaque région critique en utilisant des méthodes d'optimisation classiques, par exemple les multiplicateurs de Lagrange et les conditions de Karush-Kuhn-Tucker. Nous distinguons trois familles de méthodes : l'optimisation multi-paramétrique linéaire (mpLP) (Gal et al., 1972), l'optimisation multi-paramétrique quadratique (mpQP) (Pistikopoulos et al., 2000) et l'optimisation multi-paramétrique non linéaire (Fiacco, 1976), qui approxime les fonctions non linéaires.

Une application majeure est de pré-calculer les solutions optimales pour tout l'espace de paramétrisation de tâche et de rechercher parmi ces solutions pendant l'inférence. La reformulation de la commande prédictive avec une fonction de coût quadratique en mpQP (Pistikopoulos et al., 2002) a mené aux *MPC on chip* (aussi appelé la commande prédictive explicite) qui a des applications, par exemple, pour le contrôle d'usine chimique (Dua et al., 2008).

Tous ces algorithmes de programmation paramétrique supposent que la fonction à optimiser est connue. Bien qu'il y ait de nombreuses situations où cette information est disponible, pour beaucoup d'applications dans le monde réel (par exemple, des simulations complexes ou des systèmes réels), la fonction à optimiser est non-linéaire, non-convexe, non-différentiable, et même boîte noire, rendant ces méthodes inapplicables.

5.3.2 Optimisation multi-tâche

Récemment, l'optimisation multi-tâche a fait l'objet d'un gain d'intérêt de la part d'une sous-communauté du calcul évolutionniste (Gupta, Y.-S. Ong et al., 2016 ; Gupta, L. Zhou et al., 2022 ; H. Zhao et al., 2023). Nous pouvons distinguer deux familles d'approches : le transfert direct ou implicite des solutions (c'est-à-dire le même espace de solutions pour toutes les tâches) et le transfert indirect ou explicite des solutions (c'est-à-dire des espaces de solutions différents pour chaque tâche). Le transfert explicite concerne des problèmes avec des tâches hétérogènes où savoir quelles informations partager entre les tâches n'est pas trivial. Les méthodes de transfert explicite utilisent de la recherche de distribution probabiliste (Gupta et Y. Ong, 2018), des vecteurs de direction de recherche (Yin et al., 2019), des heuristiques d'ordre supérieur (Hao et al., 2021), ou des modèles substituts (A. T. W. Min et al., 2019).

Les algorithmes d'optimisation multi-tâche sont utilisés dans de nombreux champs applicatifs mais ne considèrent que rarement plus d'une dizaine de tâches : 3 morphologies pour des contrôleurs de robot utilisant de la neuro-évolution (Bali, Y.-S. Ong et al., 2020) ; 2 jeux de données pour de la régression symbolique en utilisant la programmation génétique (Zhong, Feng et al., 2020) ; 10 fonctions de calcul numérique en C pour le test de logiciel ou 3 modèles de voitures pour la conception utilisant un algorithme d'évolution multi-factorielle (Sagarna et al., 2016 ; Yokoya et al., 2019) ; 2 tâches auxiliaires et une tâche principale pour la planification de chemin multi-UAV (Bali, Gupta et al., 2021) ; 2 systèmes de répartition d'énergie à optimiser en utilisant un algorithme d'évolution multi-objectif multi-factoriel (J. Liu et al., 2020) ; 4 problèmes de planification de livraison en utilisant une méthode de transfert explicite de connaissance (Feng, Y. Huang et al., 2021) ; 3 modèles de diode à concevoir en utilisant une méthode de calcul évolutionniste multi-tâche guidée par la similarité (J. Liang et al., 2020).

D'autres champs de recherche, tels que l'optimisation bayésienne (Shahriari et al., 2016 ; Brochu et al., 2010), proposent des algorithmes multi-tâche, par exemple, pour ajuster des algorithmes d'apprentissage pour différents jeux de données (Pearce et al., 2018). Leurs applications sont plus focalisées sur des fonctions de *fitness* coûteuses à évaluer et peuvent difficilement supporter plus d'un millier d'évaluations en raison du coût cubique des processus gaussiens.

Peu de travaux cherchent à résoudre de nombreuses tâches (c'est-à-dire plus d'une dizaine) : 30 pour Liaw et al. (2017) et Liaw et al. (2019), 50 pour Z. Liang et al. (2022), 500 pour C. Wang et al. (2022), 2 000 pour Huynh Thi Thanh et al. (2023b), et 5 000 pour *Multi-Task MAP-Elites* (Mouret et Maguire, 2020).

5.3.3 *Multi-Task MAP-Elites*

Puisant son origine dans la recherche de diversité comportementale (Lehman et al., 2011a), *Multi-dimensional Archive of Phenotypic Elites* (MAP-Elites) (Mouret et Clune, 2015) (Section 2.5.1.2) trouve un large et divers ensemble de solutions pour une tâche. *Multi-Task MAP-Elites* (MT-ME) (Mouret et Maguire, 2020) (Section 4.2.1) reprend les mêmes principes algorithmiques mais remplace l'espace des comportements par l'espace des tâches. Tout comme

MAP-Elites, MT-ME construit une archive comme un nombre fini de cellules, chacune correspondant à une tâche, dans le but d'itérativement remplir chaque cellule avec une solution de plus haute qualité. La première étape de MT-ME est donc de discrétiser l'espace des tâches.

Une première limitation de cette discrétisation est que MT-ME ne peut ainsi résoudre qu'une sous-partie finie de l'espace des tâches bien que l'espace soit continu. Une seconde limitation est qu'étant un algorithme élitiste, son efficacité à trouver des solutions est corrélée à sa pression de sélection, qui est elle-même inversement corrélée au nombre de cellules dans l'archive. Autrement dit, plus l'archive a de cellules à remplir, plus il est facile pour une mauvaise solution de devenir une élite, et plus l'algorithme prend du temps à trouver pour chaque tâche une solution de qualité. Comme MT-ME définit le nombre de cellules dans l'archive égal au nombre de tâches à résoudre (c'est-à-dire une cellule par tâche), plus il y a de tâches à résoudre, plus il prendra de temps à trouver des solutions de qualité.

5.3.4 L'apprentissage par renforcement profond

Comme présenté dans la section 5.2, le produit final d'un algorithme d'optimisation de tâches paramétriques doit être une fonction $G : \Theta \to \mathcal{X}$ qui prend en entrée une paramétrisation de tâche θ et qui renvoie la solution optimale correspondante x^*_θ. Cette fonction est similaire à une politique d'action rencontrée en apprentissage par renforcement profond, $\Pi : S \to A$ qui prend en entrée l'état courant $s \in S$ et qui renvoie l'action optimale correspondante $a \in A$ du point de vue d'une récompense à long terme $r \in \mathbb{R}$. Nous pouvons voir la similarité entre la paramétrisation de la tâche θ et l'état s, la variable x et l'action a, et la *fitness* f et la récompense r.

La différence principale est que les algorithmes d'optimisation de tâches paramétriques optimisent directement la fonction de *fitness*. En revanche, les algorithmes d'apprentissage par renforcement résolvent un problème plus difficile dès lors qu'ils doivent optimiser une action pour une récompense à long terme, ce qui nécessite de résoudre le problème d'assignation du crédit. Néanmoins, un algorithme d'apprentissage par renforcement profond appliqué avec un horizon d'un seul pas résout un problème d'optimisation de tâches paramétriques.

Nous faisons part de ce parallèle pour mettre en lumière que, tout comme l'apprentissage par renforcement aujourd'hui, l'optimisation de tâches paramétriques pourrait avoir des applications dans de nombreux domaines et aussi qu'un algorithme d'apprentissage par renforcement est une méthode de référence qui a du sens pour notre problème. Nous avons de nouveau choisi PPO (Schulman, Wolski et al., 2017) (Section 2.3.2.1) comme méthode de référence pour l'apprentissage par renforcement profond.

5.4 Méthode

Parametric-Task MAP-Elites (PT-ME) est basé sur les mêmes principes que MT-ME. Après avoir initialisé chaque cellule de l'archive avec une élite aléatoire (Alg.2, `Line`.19), la boucle principale consiste à générer une nouvelle solution candidate, à l'évaluer pour une nouvelle tâche (`L`.41) et à mettre à jour l'archive en remplaçant l'élite par la nouvelle solution si celle-ci a une *fitness* supérieure ou égale (`L`.43-46). La première différence est que l'algorithme garde en mémoire chaque évaluation (`L`.42) qui seront utilisées pour de la distillation (Section 5.4.5) et l'évaluation (Section 5.5.2.3). La seconde différence est que PT-ME utilise l'opérateur de variation de MT-ME uniquement pour la moitié des itérations (Section 5.4.2) et utilise un nouvel opérateur pour l'autre moitié (Section 5.4.3).

5.4.1 L'archive d'élites

Comme MT-ME, PT-ME commence par diviser l'espace de paramétrisation des tâches en différentes régions à l'aide de CVT (Q. Du et al., 1999) (L.13) pour obtenir un ensemble de centroïdes qui recouvrent l'espace des tâches de façon homogène et attribue une cellule de l'archive à chaque centroïde θ_c (L.16-21). La différence principale est qu'au lieu d'évaluer les solutions candidates uniquement sur ces centroïdes, PT-ME échantillonne une nouvelle paramétrisation de tâche θ à chaque itération (L.29-31 et L.35) et l'assigne à la cellule du plus proche centroïde θ_c (L.32 et L.36). Ce faisant, PT-ME améliore la couverture de l'espace de paramétrisation des tâches à chaque itération et profite directement d'un plus large budget d'évaluations sans réduire la vitesse de convergence vers des solutions de qualité en début d'exécution. Un potentiel désavantage (que nous n'avons pas observé lors de nos expériences) est que des solutions de tâches proches peuvent être en compétition pour la même cellule de l'archive. Cela signifie qu'une solution pour une tâche "facile" peut devenir l'élite de la cellule et ainsi empêcher l'algorithme de trouver une solution pour une tâche plus "difficile" rattachée à la même cellule. Cet effet est facilement atténué en augmentant le nombre de cellules, ce qui diminue la distance maximale entre deux tâches d'une même cellule.

L'impact du nombre de cellules dans l'archive reste principalement la pression de sélection sur les élites. Aux extrêmes, avec une cellule, la pression de sélection est maximale et chaque solution est en compétition avec toutes les autres, et avec une infinité de cellules, la pression de sélection est nulle et chaque solution devient sa propre élite (du fait que chaque paramétrisation de tâche n'est rencontrée qu'une seule fois). L'impact secondaire du nombre de cellules concerne le nouvel opérateur de variation et est discuté dans la section 5.4.3. Nous avons trouvé empiriquement qu'une valeur de $N = 200$ cellules est une valeur robuste pour les différents problèmes présentés dans la section 5.5.1.

5.4.2 Opérateur de variation n°1 : SBX avec tournoi

De façon similaire à MT-ME (mais uniquement pour la moitié des itérations), PT-ME utilise un tournoi afin de biaiser la tâche sélectionnée pour évaluer la solution candidate. La différence étant qu'il échantillonne uniformément les tâches candidates dans l'espace de paramétrisation des tâches au lieu de choisir aléatoirement parmi l'ensemble fini des centroïdes formant l'archive.

Plus formellement, pour la moitié des itérations (L.28-33), PT-ME procède comme suit : sélectionne aléatoirement deux élites parents de l'archive, p_1 et p_2 (L.28) ; échantillonne s paramétrisations de tâches $\theta_{1:s}$ (L.29) ; sélectionne la paramétrisation la plus proche de celle sur laquelle a été évalué le premier parent p_1 (L.31) ; et utilise SBX (Agrawal et al., 2000) pour obtenir une solution candidate x (L.33). L'intuition est que deux tâches proches ont plus de chances de partager des solutions, mais que toujours prendre la tâche la plus proche ne permet pas d'explorer.

Comme MT-ME, PT-ME utilise un bandit pour choisir la taille s du tournoi (L.49-50), avec une taille minimale de 1 (c'est-à-dire pas de tournoi) et une taille maximale de 500 (c'est-à-dire choisir une tâche proche). PT-ME utilise aussi l'algorithme de bandit UCB1 (Auer et al., 2002), qui obtient un regret optimal à un coefficient multiplicatif près. Comme MT-ME, le score du bandit correspond aussi au fait que la solution candidate devienne une élite (L.47-48).

Dans une étude préliminaire, nous avons conclu qu'utiliser un bandit améliore significativement les performances par rapport à choisir la taille du tournoi aléatoirement, mais que c'est légèrement moins bien que choisir la taille du tournoi à la main pour chaque problème. Nous avons décidé de garder le bandit étant donné que cela enlève le besoin d'ajuster un hyper-paramètre sans une perte significative de performances.

Nous avons aussi comparé les opérateurs de variations SBX (Agrawal et al., 2000) et *iso-line* (Vassiliades et Mouret, 2018) et avons conclu qu'il n'y a pas de différence significative. Nous avons donc décidé d'utiliser SBX, qui est plus communément utilisé.

5.4.3 Opérateur de variation n°2 : Régression linéaire locale

SBX n'exploite pas la structure multi-tâche du problème étant donné qu'il n'utilise pas le fait que la solution sur laquelle sera évaluée la solution candidate est connue. En s'inspirant du tournoi, mais au lieu de biaiser le choix de la tâche étant donné les parents de la solution candidate, nous proposons de biaiser le choix de la solution candidate étant donnée la tâche sélectionnée. L'idée est d'utiliser les élites pour estimer la fonction $G : \theta \mapsto x^*_\theta$. Pour ce faire, nous proposons d'agréger les informations de l'archive courante en construisant un modèle local de la fonction G et de l'utiliser pour estimer une solution candidate. Nous avons choisi un modèle linéaire pour proposer une solution candidate car il est simple et rapide.

Pour la moitié des itérations (L.35-40), PT-ME :

1. échantillonne une paramétrisation de tâche θ (L.35) ;
2. trouve le plus proche centroïde θ_c (L.36) en utilisant un KDTree (Maneewongvatana et al., 1999) pré-calculé à la création de l'archive à l'aide de l'implémentation SciPy (Virtanen et al., 2020) (L.15) ;
3. trouve les cellules adjacentes $A[\theta_c].adj$ (en utilisant une triangulation de Delaunay (Delaunay, 1928) pré-calculée à la création de l'archive à l'aide de l'implémentation SciPy (Virtanen et al., 2020) (L.14)) et extrait leur paramétrisation de tâche $\boldsymbol{\theta}$ et solutions $\boldsymbol{x}$ (L.37-38) ;
4. réalise une méthode des moindres carrés linéaire, $M = (\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\theta})^{-1}\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x}$ (L.39) ;
5. calcule la solution candidate en ajoutant un bruit additionnel (L.40), $x = M \cdot \theta + \sigma_{reg} \cdot \mathcal{N}\big(0, \text{variance}(\boldsymbol{x})\big)$ (où σ_{reg} régule l'intensité du bruit gaussien.)

Nous utilisons un critère d'adjacence (c'est-à-dire la triangulation de Delaunay) et non pas de distance (par exemple, les k plus proches voisins ou les voisins plus proches que ϵ). Cela permet d'adapter le nombre de cellules adjacentes à la dimension, enlevant ainsi la nécessité d'ajuster un hyper-paramètre (c'est-à-dire le nombre de voisins k ou la distance ϵ). Le coefficient du bruit ajouté après la régression, σ_{reg}, équilibre exploration et exploitation. Lors d'expériences préliminaires, nous nous sommes rendus compte que différents problèmes nécessitaient différentes valeurs optimales. Nous avons convergé sur le choix d'une valeur standard de 1, bien qu'ajuster finement cette valeur pour chaque problème ou mieux, avoir une méthode qui adapte ce paramètre pourrait accroître les performances.

5.4.4 Algorithme

L'algorithme 2 détaille le pseudo-code de PT-ME. Les modifications majeures par rapport à MT-ME (Mouret et Maguire, 2020) sont surlignées en couleurs et résumées en :

— un échantillonnage uniforme à chaque itération d'une nouvelle paramétrisation de tâche pour évaluer la solution candidate ;

— un nouvel opérateur de variation qui utilise une régression linéaire locale utilisée à 50% des itérations.

Nous avons aussi effectué deux modifications mineures. Premièrement, nous avons remplacé la condition de supériorité stricte pour devenir élite par une condition de supériorité non stricte. En permettant à l'algorithme de remplacer plus souvent une élite par une autre de *fitness* identique, il peut tester plus de solutions différentes et donc améliorer son exploration. En pratique, pour les deux problèmes jouets présentés dans la section 5.5.1, ce changement apporte une légère amélioration des performances, mais non significative. Ce qui peut s'expliquer par le fait que dans le cas de fonctions de *fitness* réelles, il y a peu de chance d'avoir deux valeurs identiques. Deuxièmement, pendant l'initialisation de l'archive, nous évaluons chaque cellule avec une solution aléatoire sur la tâche représentée par son centroïde θ_c au lieu d'évaluer un nombre arbitraire de solutions aléatoires sur des cellules, elles aussi, tirées aléatoirement. Ceci permet de retirer un hyper-paramètre et d'avoir directement une élite dans chaque cellule après initialisation.

5.4.5 Généralisation

Avec un budget fini d'évaluations, l'algorithme ne peut résoudre qu'un nombre fini de tâches. Afin de résoudre le problème d'optimisation de tâches paramétriques et proposer une solution pour chaque paramétrisation de tâche, l'algorithme doit renvoyer une approximation $\hat{G}$ de la fonction $G : \theta \mapsto x^*_\theta$.

Nous proposons trois méthodes permettant de généraliser le jeu de données renvoyé par PT-ME afin d'obtenir une approximation $\hat{G}$:

— **PT-ME Interpolation** interpole les élites de l'archive en réutilisant l'opérateur de variation de régression linéaire locale (Section 5.4.5.1) ;
— **PT-ME Interpolation Distillation** est une distillation de cet opérateur dans un réseau de neurones (Section 5.4.5.2) ;
— **PT-ME Direct Distillation** est une distillation directe des solutions de l'archive dans un réseau de neurones (Section 5.4.5.3).

5.4.5.1 Interpolation des élites

Pour interpoler les élites d'une archive, une première méthode est d'utiliser simplement une version modifiée de l'opérateur de variation de régression linéaire locale en enlevant le bruit gaussien.

Plus formellement, pour une nouvelle tâche θ, **PT-ME Interpolation** :

1. trouve le plus proche centroïde θ_c (à l'aide d'un KDTree) ;
2. trouve les cellules adjacentes $A[\theta_c].adj$ (à l'aide d'une triangulation de Delaunay) ;
3. extrait leur paramétrisation de tâche $\boldsymbol{\theta}$ et solutions $\boldsymbol{x}$;
4. réalise une méthode des moindres carrés linéaire, $M = (\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\theta})^{-1}\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x}$;
5. calcule la solution correspondante $x = M \cdot \theta$.

Cette méthode a pour avantage de ne pas nécessiter d'apprentissage supplémentaire.

Algorithm 2 Parametric-Task MAP-Elites
Contributions : Parametric-Task New variation operator

Parameters :
d_x : dimension of the solution space $\mathcal{X} = [0,1]^{d_x}$
d_θ : dimension of the task parameter space $\Theta = [0,1]^{d_\theta}$
`fitness` : $\mathcal{X} \times \Theta \to \mathbb{R}$
Hyper-parameters :
B : budget of evaluations (i.e., 100 000)
N : number of cells (i.e., 200)
S : possible tournament sizes (i.e., $[1, 5, 10, 50, 100, 500]$)
σ_{SBX} : mutation factor for SBX (Agrawal et al., 2000) (i.e., 10.0)
σ_{reg} : mutation factor for our new variation operator (i.e., 1.0)
Initialization :
$E \leftarrow \emptyset$ ▷ E : to store all the evaluations
$C \leftarrow N$ random centroids using CVT ▷ Divise Θ into cells
$D \leftarrow \text{DelaunayTriangulation}(C)$ ▷ For computing adjacency
$kdtree \leftarrow \text{KDTree}(C)$ ▷ To quickly find the cell of a task
$A \leftarrow \emptyset$ ▷ A : Archive
for θ_c `in` C **do** ▷ one for each cell
    $A[\theta_c].\theta = \theta_c$ ▷ θ : the task parameter
    $A[\theta_c].x = \texttt{random.uniform}(0, 1, d_x)$ ▷ x : the solution
    $A[\theta_c].f = \texttt{fitness}(A[\theta_c].x, A[\theta_c].\theta)$ ▷ f : the fitness
    $A[\theta_c].adj = D[\theta_c]$ ▷ adj : the adjacent cells
Main loop :
$s \leftarrow$ random_in_list(S) ▷ s : tournament size
$selected \leftarrow \text{zeros}(\text{len}(S))$ ▷ Counter of selection for each size
$successes \leftarrow \text{zeros}(\text{len}(S))$ ▷ Counter of successes for each size
for $\_ = 0$ `to` $B - N$ **do** ▷ We already consumed N evaluations
    if random() ≤ 0.5 **then** ▷ Use SBX with tournament (50%)
        $p1, p2 \leftarrow$ two elites randomly selected from A
        $\theta_{1:s} \leftarrow \text{random.uniform}(0, 1, (s, d_\theta))$ ▷ Candidate tasks
        $selected[s] \leftarrow selected[s] + 1$ ▷ Update the bandit
        $\theta \leftarrow \text{closest}(\theta_{1:s}, p1.\theta)$ ▷ Task parameters tournament
        $\theta_c \leftarrow \texttt{kdtree.find}(\theta)$ ▷ Find the corresponding cell
        $x \leftarrow \text{SBX}(p1, p2, \sigma_{SBX})$ ▷ Mutation & Cross-over
    else ▷ Use new variation operator : Local Linear Regression (50%)
        $\theta \leftarrow \text{random.uniform}(0, 1, d_\theta)$
        $\theta_c \leftarrow \texttt{kdtree.find}(\theta)$ ▷ Find the corresponding cell
        $\boldsymbol{\theta} \leftarrow \{A[\theta_i].\theta\}_{\theta_i \in A[\theta_c].adj}$
        $\boldsymbol{x} \leftarrow \{A[\theta_i].x\}_{\theta_i \in A[\theta_c].adj}$
        $M \leftarrow (\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\theta})^{-1}\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x}$ ▷ Linear least squares
        $x \leftarrow M \cdot \theta + \sigma_{reg} \cdot \mathcal{N}(0, \text{variance}(\boldsymbol{x}))$ ▷ Regression + noise
    $f \leftarrow \texttt{fitness}(x, \theta)$ ▷ Evaluate
    $E.\text{append}((\theta, x, f))$ ▷ Store the evaluation
    if $f \geq A[\theta_c].f$ **then** ▷ Update the archive
        $A[\theta_c].\theta = \theta$
        $A[\theta_c].x = x$
        $A[\theta_c].f = f$
        if tournament was used **then** ▷ Update the bandit
            $successes[s] \leftarrow successes[s] + 1$
    if tournament was used **then** ▷ Update the tournament size
        $s \leftarrow S\left[\underset{j}{\text{argmax}}\left(\frac{successes[j]}{selected[j]} + \sqrt{\frac{2\ln(\text{sum}(selected))}{selected[j]}}\right)\right]$
return E

Algorithm 3 Calcule une archive avec une nouvelle résolution en partant du jeu de donnée de solutions

Parameters :
d_θ : dimension of the task parameter space $\Theta = [0,1]^{d_\theta}$
E : list of evaluations (θ, x, f)
N : number of cells in the new archive
Initialization :
$C \leftarrow N$ random centroids using CVT
$kdtree \leftarrow \text{KDTree}(C)$
$A \leftarrow \{\theta_c : \{\theta : None, x : None, f : 0\}$ `for` θ_c `in` $C\}$
Main loop :
for (θ, x, f) `in` E **do** ▷ Compute elites
    $\theta_c \leftarrow$ `kdtree.find`(θ) ▷ Find the corresponding cell
    if $f \geq A[\theta_c].f$ **then** ▷ We suppose f normalized so that $f \geq 0$
        $A[c_i] = (\theta, x, f)$
return A

5.4.5.2 Interpolation des élites puis distillation

Une seconde méthode qui se rapproche davantage d'une politique d'apprentissage par renforcement profond consiste à distiller l'interpolation des élites dans un réseau de neurones (avec une architecture identique à celle de l'implémentation de *stable-baselines* (Raffin et al., 2021) de PPO (Schulman, Wolski et al., 2017), c'est-à-dire deux couches cachées avec 64 neurones et des fonctions d'activation ReLU).

Plus formellement, l'algorithme d'apprentissage :

1. échantillonne 10 000 tâches réparties uniformément dans l'espace de paramétrisation des tâches Θ en utilisant CVT (Q. Du et al., 1999) ;
2. utilise l'interpolation présentée dans la section 5.4.5.1 pour obtenir une solution pour chacune de ces tâches ;
3. effectue un apprentissage supervisé avec ce jeu de données de 10 000 paires (tâche et solution) pour entraîner le réseau de neurones en utilisant de la régression avec une fonction de coût des moindres carrés moyens.

L'intérêt de cette méthode est d'obtenir à la fin une méthode rapide pour obtenir une bonne solution à une nouvelle tâche.

5.4.5.3 Distillation directe

Une troisième méthode consiste à effectuer une distillation directe en entraînant directement le réseau de neurones avec les élites d'une archive recalculée pour une résolution arbitraire grâce à l'algorithme 3. L'avantage est qu'elle apprend à partir de données réelles et non pas interpolées. Le désavantage est que l'apprentissage supervisé nécessite une quantité importante de données. Cela peut poser problème, car plus l'archive contient de cellules (et donc d'élites) moins les élites sont de bonne qualité. Ceci peut être contrebalancé en augmentant le budget d'évaluations de PT-ME pour augmenter la densité des bonnes solutions.

5.5 Expérimentations

5.5.1 Problèmes considérés

5.5.1.1 *10-DoF Arm*

C'est le principal problème jouet présenté dans MT-ME (Mouret et Maguire, 2020) et réutilisé dans C. Wang et al. (2022) et Huynh Thi Thanh et al. (2023a). Il consiste à trouver la position des articulations pour plusieurs morphologies d'un bras articulé avec 10 degrés de liberté qui place l'organe terminal aussi proche que possible d'une cible fixée (Figure 5.1.a).

L'espace de paramétrisation des tâches Θ correspond à l'angle maximal $\alpha_{\max}$ et à la longueur L de chaque articulation (toutes les articulations ayant la même taille). La fonction de *fitness* correspond à la distance euclidienne entre l'organe terminal à la position pos_{ee} et la cible fixée à la position pos_{tar}. Afin d'obtenir un problème de maximisation, nous passons cette distance dans une exponentielle négative : $f = \exp(-||pos_{ee} - pos_{tar}||^2)$ de sorte qu'elle appartienne à l'intervalle [0, 1]. L'espace de solution $\mathcal{X}$ correspond à la position angulaire de chaque articulation (normalisée par sa position maximale $\alpha_{\max}$).

5.5.1.2 *Archery*

Ce problème jouet correspond à tirer une flèche sur une cible éloignée et à calculer le score associé (Figure 5.2.a).

L'espace de paramétrisation des tâches Θ correspond à la distance D à la cible (entre 5 m et 40 m), et à la force et l'orientation d'un vent horizontal, simulé par une accélération additionnelle constante (entre -10m.s^{-2} et 10m.s^{-2}). L'espace de solution $\mathcal{X}$ correspond au tangage et au lacet de la flèche tirée (entre $-\frac{\pi}{12}$ rad et $\frac{\pi}{12}$ rad) avec une vélocité constante v de 70 m.s^{-1}. L'espace de solution $\mathcal{X}$ n'a que deux dimensions, mais la fonction de *fitness* est éparse car elle est nulle lorsque la flèche n'atteint pas la cible, rendant le problème difficile.

La fonction de fitness f correspond au score traditionnel du tir à l'arc normalisé entre 0 et 1. Plus formellement, la fonction calcule la vitesse

$$\vec{v} = \begin{pmatrix} -\sin(yaw) \\ \cos(yaw)\cos(pitch) \\ \cos(yaw)\sin(pitch) \end{pmatrix},$$

déduit l'instant d'impact avec la cible $t = \frac{D}{v_y}$, calcule la distance au centre de la cible $d = ||\frac{1}{2}(-g\vec{z} + W\vec{x})t^2 + \vec{v}t||^2$ et calcule finalement la *fitness* $f = \max(0, \text{int}(10 - \frac{d}{0{,}061}))/10$.

5.5.1.3 *Door-Pulling*

Un robot humanoïde simulé doit ouvrir une porte en tirant sur une poignée verticale (Figure 5.3.a). Ce problème est plus similaire à un problème d'apprentissage par renforcement. Cependant, au lieu d'entraîner une politique pour tirer la poignée étant donné un état initial aléatoire, nous optimisons explicitement pour trouver une commande de haut niveau qui permet d'ouvrir la porte pour chaque configuration possible de la porte autour du robot.

L'espace de paramétrisation des tâches Θ correspond à la position de la porte par rapport au robot (entre 0,8 m et 1,2 m devant le robot et entre 0 m et -0,5 m sur le côté) et à son orientation (entre $\frac{3\pi}{8}$ rad et $\frac{5\pi}{8}$ rad). La fonction de *fitness* correspond à l'angle de la porte après ouverture. C'est une *fitness* éparse car l'angle de la porte reste nul si le robot n'arrive pas à saisir la poignée.

L'espace de solution $\mathcal{X}$ correspond aux commandes de haut niveau suivantes envoyées au contrôleur du corps complet :

— la translation de la pince du robot par rapport à la poignée (3 degrés de liberté) ;
— l'orientation de la pince avec un tangage constant (2 degrés de liberté) ;
— la translation de la pince pendant le tirage (3 degrés de liberté) ;
— la rotation verticale de la pince pendant le tirage (1 degré de liberté).

Le contrôleur du corps complet résout à 1 000 Hz un problème quadratique qui optimise les positions articulaires du robot pour satisfaire la dynamique quasi-statique tout en essayant de satisfaire au mieux les commandes de haut-niveau (Dalin et al., 2021).

5.5.2 Méthodologie

5.5.2.1 Méthodes de référence

— **Random** échantillonne uniformément la paramétrisation de tâche θ et la solution candidate x (elle répond à la question : "Quelle est la difficulté du problème ?") ;
— **CMA-ES** échantillonne uniformément la paramétrisation de tâche θ, utilise CMA-ES (Hansen et al., 2003) jusqu'à atteindre l'un de ses critères de terminaison, et itère jusqu'à atteindre le budget B d'évaluations ("Quelle est la performance de CMA-ES ?") ;
— **CMA-ES (max 10)** fait la même chose mais limite CMA-ES à 10 itérations par paramétrisation de tâche ("Est-ce que CMA-ES gagne à visiter plus de tâches ?") ;
— **PPO** exécute l'implémentation *stable-baseline* (Raffin et al., 2021) de PPO (Schulman, Wolski et al., 2017) avec un échantillonnage uniforme de la paramétrisation de tâche comme état initial, la solution x comme action et la *fitness* comme récompense ("Comment PT-ME se compare-t-il face à une méthode d'apprentissage par renforcement profond ?") ;
— **MT-ME** exécute MT-ME avec 5 000 tâches échantillonnées avec CVT.

5.5.2.2 Les ablations

— **PT-ME** exécute l'algorithme 2 ;
— **PT-ME (sans régression)** exécute l'algorithme 2 avec uniquement SBX et le tournoi ("Notre nouvel opérateur de variation est-il utile ?") ;
— **PT-ME (100% régression)** exécute l'algorithme 2 avec uniquement notre nouvel opérateur de variation ("Notre nouvel opérateur de variation est-il suffisant ?") ;
— **PT-ME (sans régression - sans tournoi)** exécute l'algorithme 2 avec uniquement SBX sans tournoi, c'est-à-dire avec une taille de tournoi s égale à 1 ("Le tournoi est-il utile ?") ;
— **PT-ME (sans tournoi)** exécute l'algorithme 2 sans le tournoi pour la partie utilisant SBX ("Le tournoi est-il toujours utile ?").

5.5.2.3 Évaluations et mesures

Après un budget B de 100 000 évaluations, nous voulons évaluer la couverture et la qualité des solutions trouvées par les différentes méthodes. L'hypothèse est qu'un algorithme optimal trouve des solutions optimales uniformément réparties sur tout l'espace des tâches. Pour ce faire, nous utilisons le QD-score (Pugh, Soros, Szerlip et al., 2015) pour évaluer les solutions

trouvées pour un ensemble fini de tâches :

$$\text{QD-Score}(A) = \frac{1}{|A|} \sum_{(\theta,x,f)_{\theta_c} \in A} f$$

où $(\theta, x, f)_{\theta_c}$ est l'élite de la cellule de centroïde θ_c.

Comme PT-ME garde en mémoire toutes les évaluations des B tâches évaluées (L.42), nous pouvons recalculer les élites pour des archives de différentes résolutions (Algorithme 3), entre une cellule (la meilleure solution trouvée sur toutes les tâches) et B cellules uniformément réparties sur l'espace de paramétrisation des tâches Θ. En effet, l'algorithme ne peut pas remplir plus de cellules que son budget d'évaluations. Nous attribuons la *fitness* minimale aux cellules vides (c'est-à-dire sans élite), ce faisant, regarder le QD-score pour l'archive de plus grande résolution ne permet pas d'évaluer complètement les performances de l'algorithme. Nous introduisons une mesure supplémentaire, le MR-QD-score (*MR-QD-Score*), qui correspond à la moyenne du QD-score pour un ensemble de résolutions de 1 à B cellules. Plus formellement :

$$\text{MR-QD-Score}(A_{1:n}) = \frac{1}{n} \sum_{i=1}^{n} \text{QD-Score}(A_i) \tag{5.2}$$

où $A_{1:n}$ sont des archives de résolutions différentes calculée à partir d'un même jeu de donnée à l'aide de l'algorithme 3. Nous utilisons $n = 50$ résolutions uniformément réparties dans l'espace logarithmique entre 1 et 100 000 cellules.

Pour simplifier la comparaison entre différents problèmes, nous normalisons la *fitness* entre sa plus mauvaise valeur (à 0) et sa valeur optimale (à 1). C'est déjà le cas pour *Archery* (c'est-à-dire pour chaque tâche, la flèche peut atteindre le centre de la cible, atteignant un score de 1 ou manquer la cible et atteindre un score minimal de 0). Pour *10-DoF Arm*, connaître les limites de la fonction de *fitness* pour chaque tâche n'est pas trivial. Pour l'estimer, nous avons exécuté CMA-ES (Hansen et al., 2003) sur chaque cellule de l'archive de plus grande résolution.

L'ensemble des comparaisons statistiques sont effectuées à l'aide d'un test de Wilcoxon-Mann-Whitney et de la correction de Bonferroni.

5.5.3 Évaluation de l'archive

5.5.3.1 10-DoF Arm

La figure 5.1 montre les résultats des différentes méthodes (20 réplications chacune) pour le problème *10-DoF Arm*.

CMA-ES alloue trop d'évaluations à la même tâche, ce qui l'empêche de couvrir l'espace et entraîne un QD-score inférieur à celui de **Random** lorsqu'il y a plus de 20 cellules à remplir. Limiter le nombre d'itérations pour une même tâche permet à **CMA-ES (10 max)** de remplir plus de cellules avec de meilleures solutions que **Random** jusqu'à une résolution de 1 000 cellules. **MT-ME** trouve des solutions de qualité similaire à celles des meilleures méthodes. Cependant, du fait qu'elle ne parcourt qu'un ensemble fixé de tâches, elle ne peut pas remplir plus de cellules que son budget, entraînant une baisse significative de son QD-score au-delà de 5 000 cellules. **PPO** et toutes les variantes de notre méthode ont un profil de QD-score similaire, réussissant à remplir la plupart des cellules avec de bonnes solutions, même à de hautes résolutions. Notre méthode **PT-ME** (de médiane $0,950$ et premier et troisième quartiles $[0,949; 0,952]$) et **PPO** ($0,950$ $[0,947; 0,953]$) ont le plus haut *MR-QD-Score* et ne sont pas significativement différents.

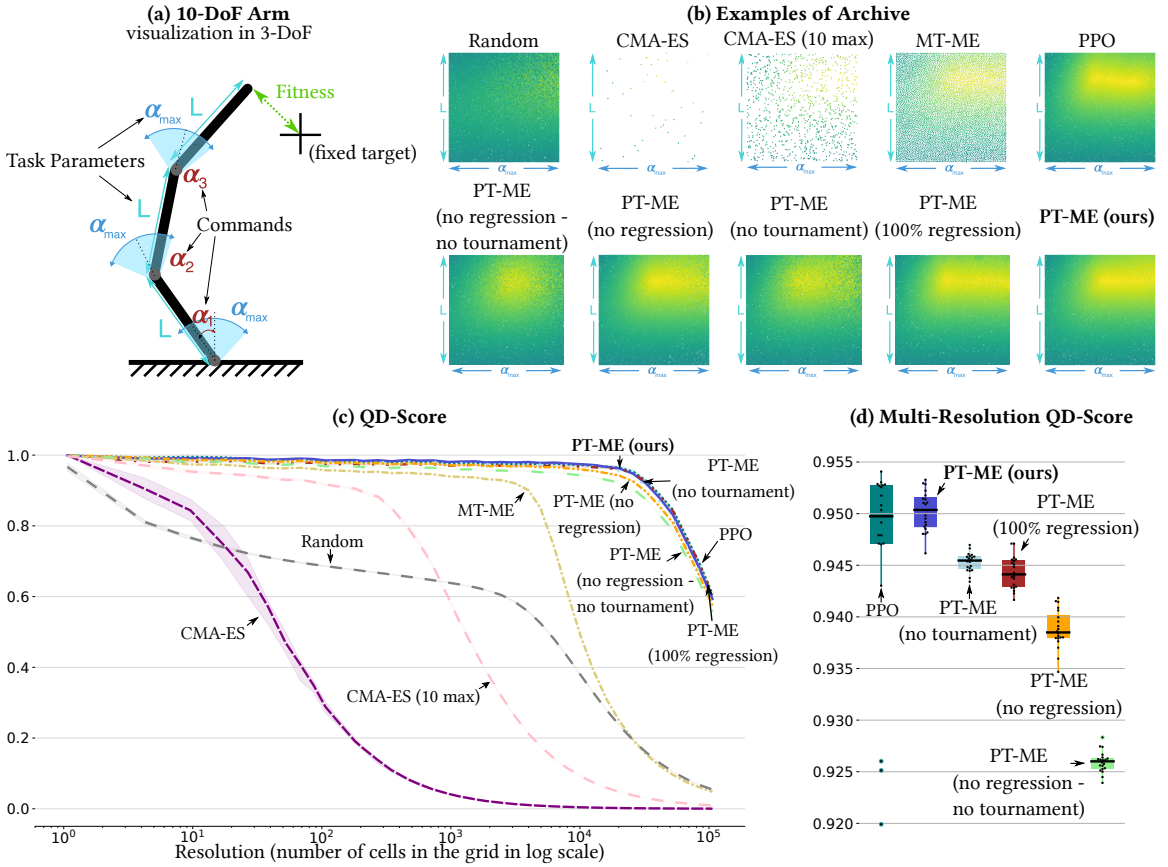

FIGURE 5.1 – (a) Vue schématique du *10-DoF Arm*. (b) Exemples d'archives extraits d'une exécution de chaque méthode. (c) Le QD-Score des différentes méthodes pour différentes résolutions d'archives. (d) Le Multi-Résolution QD-Score correspondant.

Le MR-QD-score de **PT-ME** est significativement supérieur (p-value $< 0,05$) à toutes les ablations, montrant ainsi que toutes les parties de l'algorithme (le passage aux tâches paramétriques, le nouvel opérateur de variation et SBX (Agrawal et al., 2000) avec un tournoi pour biaiser la sélection de la tâche) sont utiles. **PT-ME (sans tournoi)** $(0,945\ [0,945; 0,946])$ et **PT-ME (sans régression - sans tournoi)** $(0,926\ [0,925; 0,926])$ sont significativement moins bons que **PT-ME** $(0,950\ [0,949; 0,952])$ et **PT-ME (sans régression)** $(0,938\ [0,938; 0,940])$, montrant ainsi que le tournoi introduit dans MT-MB (Mouret et Maguire, 2020) est bénéfique.

La figure 5.1.b montre des archives tirées de l'exécution des différentes méthodes. **CMA-ES** et **MT-ME** ont des archives éparses puisqu'elles allouent plusieurs évaluations par tâche. **Random** couvre tout l'espace de paramétrisation des tâches mais ne trouve pas de solutions aussi bonnes que **PPO** et **PT-ME** et ses variantes.

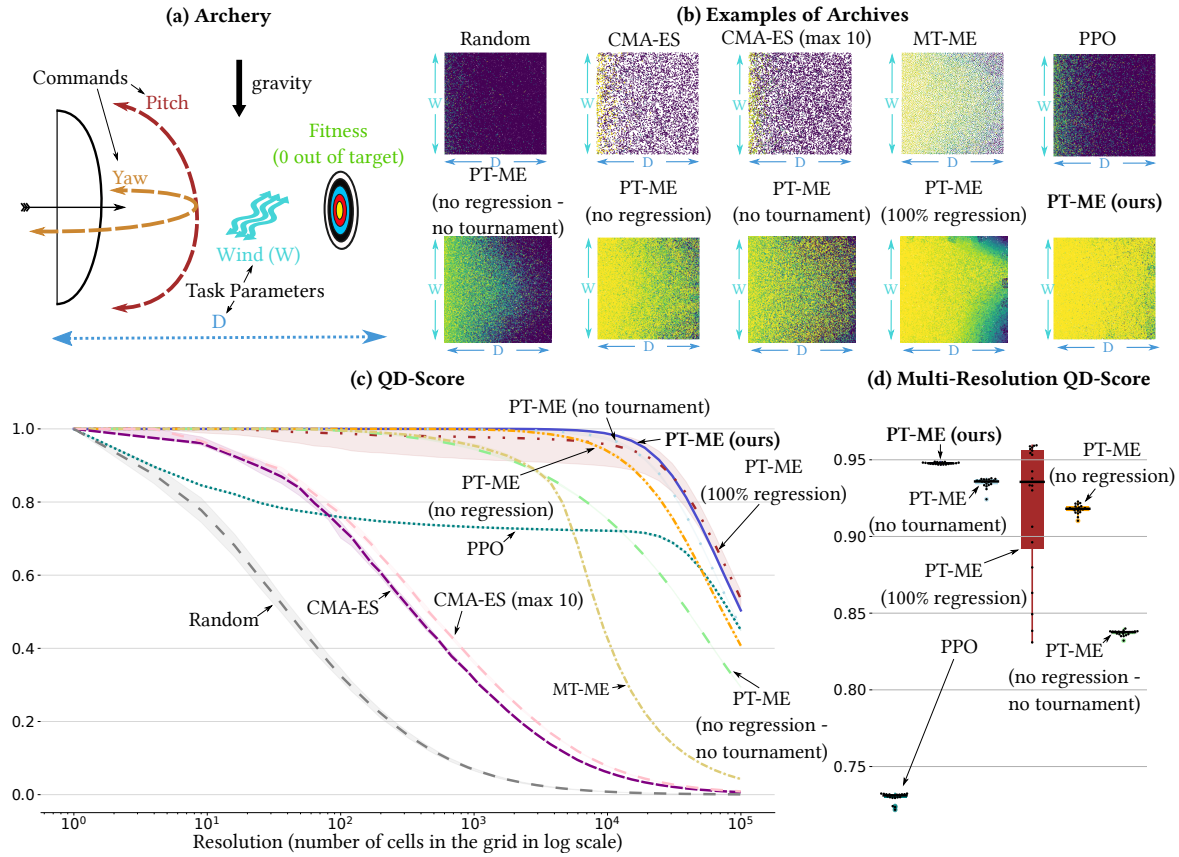

FIGURE 5.2 – (a) Vue schématique de *Archery*. (b) Exemples d'archives extraites d'une exécution de chaque méthode. (c) Le QD-Score des différentes méthodes pour différentes résolutions d'archives. (d) Le Multi-Résolution QD-Score correspondant.

5.5.3.2 Archery

Nous comparons ensuite les différentes méthodes (20 réplications pour chacune) sur le problème *Archery* (Figure 5.2.c). Comparé à *10-DoF Arm*, où chaque solution candidate atteint une *fitness* non nulle, pour *Archery*, la plupart de l'espace des solutions (c'est-à-dire approximativement 95%) ont une *fitness* nulle, entraînant la plus mauvaise performance pour **Random**.

CMA-ES et **CMA-ES (10 max)** ont des profils de QD-Score similaires, car soit la première génération de solution générée par CMA-ES est nulle et l'algorithme s'arrête, soit l'une des solutions candidates atteint par chance la cible, et CMA-ES trouve rapidement (c'est-à-dire en moins de 10 itérations) la solution optimale.

MT-ME $0,726$ $[0,724; 0,728]$ surpasse **CMA-ES** $0,392$ $[0,384; 0,398]$ et **CMA-ES (10 max)** $0,416$ $[0,414; 0,420]$, montrant encore que résoudre toutes les tâches simultanément est plus efficace que de les résoudre séparément. Pour des résolutions inférieures à 10 000 cellules, **MT-ME** est aussi meilleure que **PPO**, bien qu'elle soit ensuite pénalisée par son ensemble fixé de tâches. **PPO** $(0,731$ $[0,730; 0,732])$ échoue pour ce problème car son exploration n'est pas assez performante pour contrebalancer la *fitness* éparse.

PT-ME $(0,948$ $[0,947; 0,948])$ est significativement meilleure que toutes les autres méthodes à l'exception de **PT-ME (100% régression)** $(0,936$ $[0,892; 0,956])$ dont la variance est significativement supérieure aux autres, du fait qu'étant une méthode purement d'exploi-

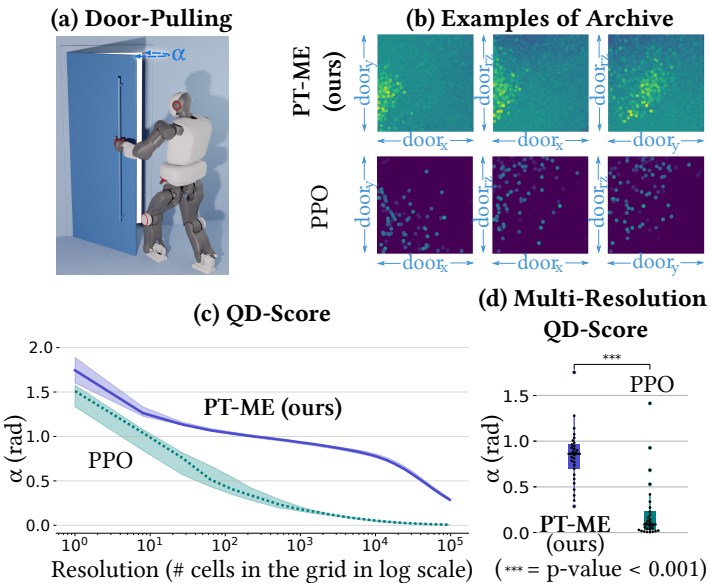

FIGURE 5.3 – (a) Le problème du robot humanoïde ouvrant une porte en tirant sur la poignée. (b) Exemples d'archives extraites d'une exécution de **PT-ME** et **PPO**. (c) Le QD-score des deux méthodes pour différentes résolutions (la courbe représente la médiane de 10 réplications et la zone colorée entre le premier et troisième quartile). (d) Le MR-QD-score des deux méthodes pour 10 réplications.

tation, elle requiert de la "chance" pour explorer tout l'espace et peut se retrouver bloquée quand les solutions courantes ne permettent pas de résoudre les régions de l'espace des tâches non encore résolues.

Figure 5.2.b présente des archives tirées de l'exécution des différentes méthodes. **CMA-ES**, **CMA-ES (10 max)** et **PPO** ne résolvent que les tâches les plus faciles (c'est-à-dire où la cible est la plus proche). **MT-ME** et les ablations de **PT-ME** ont des profils d'archive similaires avec de mauvaises solutions dans les coins où la cible est la plus loin et le vent le plus fort.

5.5.3.3 Door-Pulling

Dû au coût de calcul pour exécuter la simulation de robot humanoïde (de l'ordre de la dizaine de secondes contre la milliseconde pour les deux problèmes jouets), nous ne comparons que notre méthode complète **PT-ME** et **PPO** sur 10 réplications.

Le résultat (Figure 5.3) montre que **PT-ME** ($0,86$ rad $[0,70; 0,96]$) surpasse significativement **PPO** ($0,09$ rad $[0,03; 0,23]$). Ceci s'explique de nouveau par la nature éparse de la *fitness* qui nécessite une bonne exploration pour d'abord atteindre la poignée et obtenir une *fitness* non nulle. L'exemple d'archive (Figure 5.3.b) montre bien que **PPO** ne réussit qu'à trouver quelques solutions par chance.

5.5.4 Évaluations de la généralisation

Pour évaluer la généralisation des différentes méthodes, nous évaluons la *fitness* des solutions proposées pour 10 000 (1 000 pour *Door-Pulling*) nouvelles tâches échantillonnées avec CVT. Par simplicité dans la publication, nous n'avons présenté que la méthode de généralisation **PT-ME Direct Distillation** (Section 5.5.4.1). Nous étendons donc dans ce chapitre aux trois différentes méthodes (Section 5.5.4.2).

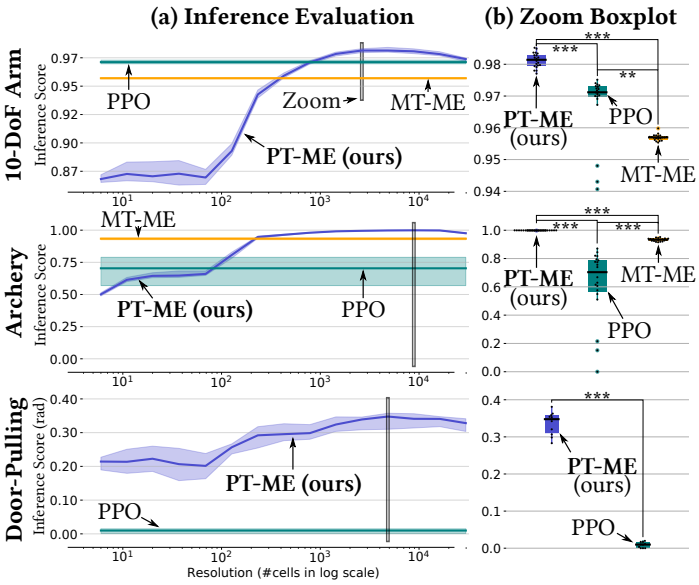

FIGURE 5.4 – (a) Score de généralisation, c'est-à-dire la *fitness* moyenne pour 10 000 (1 000 pour *Door Pulling*) nouvelles tâches uniformément réparties de la distillation de **PT-ME**, la distillation de **MT-ME** et **PPO** pour les trois problèmes d'optimisation de tâches paramétriques. La ligne en trait plein correspond à la médiane de 20 réplications (10 pour *Door Pulling*) et la zone en couleur entre le premier et le troisième quartile. (b) Boîtes à moustaches des différentes méthodes avec la meilleure résolution pour la distillation de **PT-ME** (comme la méthode nous permet de choisir la résolution). (***= p-value $< 0,001$, **= p-value $< 0,01$)

5.5.4.1 Évaluation principale

La figure 5.4 compare la capacité de généralisation de distillations directes des solutions de **PT-ME** pour différentes résolutions d'archive, une distillation directe des élites de **MT-ME** et **PPO**. **PT-ME** surpasse significativement **PPO** sur chaque problème (p-value $< 0,001$).

Le score d'inférence de **PT-ME** commence par augmenter avec la résolution de l'archive, montrant ainsi le besoin d'un large jeu de données pour la distillation. Puis, ce score diminue pour de fortes résolutions (c'est-à-dire pour plus de 10 000 cellules) à cause de la baisse de qualité des élites due à la baisse de la pression de sélection (plus il y a de cellules dans l'archive, plus une mauvaise solution a de chances de devenir une élite). Exécuter l'algorithme pour un budget plus important devrait augmenter la résolution maximale qui permet de conserver des élites de haute qualité, bien que pour *Archery*, **PT-ME** atteint un score presque parfait (la plus mauvaise exécution a un score d'inférence de $0,997$ sur 1).

Pour les deux problèmes jouets, **PT-ME** surpasse **MT-ME** pour des résolutions supérieures à 500 cellules et atteint des scores presque parfaits. Ceci montre que **PT-ME** couvre l'espace des solutions avec de meilleures solutions et résout effectivement le problème d'optimisation de tâche paramétrique.

Pour *Door-Pulling*, le score de généralisation est significativement moins bon que durant l'optimisation ($0,47$ $[0,44; 0,49]$ contre $0,86$ $[0,70; 0,96]$). De plus, la politique distillée est encore moins bonne ($0,35$ $[0,33; 0,41]$), même si elle est toujours meilleure que **PPO**, qui ne réussit toujours pas à résoudre ce problème ($0,01$ $[0,00; 0,02]$). Ceci peut s'expliquer par la nature chaotique de la dynamique de contact dans notre simulateur.

5.5.4.2 Évaluation étendue

La figure 5.5 présente une comparaison de l'ensemble des méthodes de généralisation. Pour l'ensemble des problèmes, les généralisations tirées de PT-ME surpassent significativement PPO.

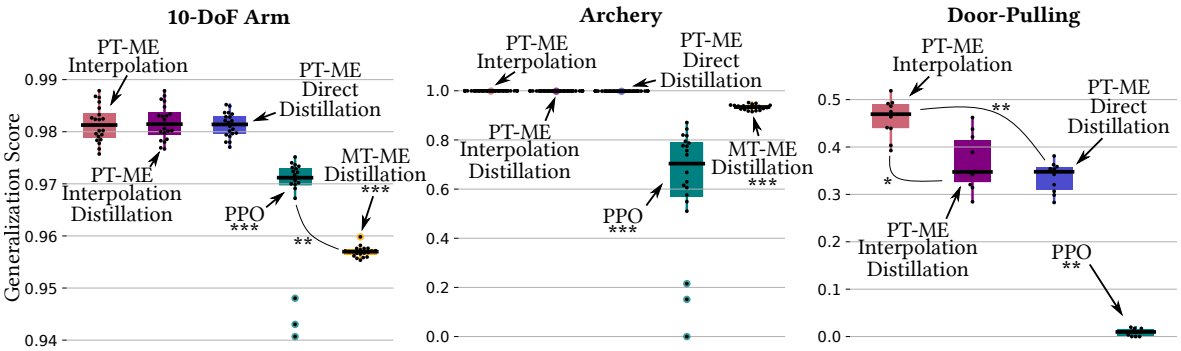

FIGURE 5.5 – Évaluation étendue de la généralisation avec différentes méthodes de généralisation du jeu de données obtenu par PT-ME sur les trois problèmes d'optimisation. Une ou des étoiles sous le nom de la méthode signifie une comparaison à toutes les autres méthodes. Une ou des étoiles avec un arc en trait plein entre deux méthodes signifie une comparaison entre ces deux méthodes ($***$= p-value $< 0,001$, $**$= p-value $< 0,01$, $*$= p-value $< 0,05$).

Pour *10-DoF Arm* et *Archery*, les trois méthodes, **PT-ME Interpolation**, **PT-ME Interpolation Distillation** et **PT-ME Direct Distillation** ont un score significativement similaire, ce qui s'explique par la densité des solutions trouvées par PT-ME. Autrement dit, il n'est pas utile d'enrichir le jeu de données d'apprentissage avec des exemples supplémentaires en utilisant l'interpolation linéaire.

Par contre, pour *Door-Pulling*, **PT-ME Interpolation** $(0,47\ [0,44;0,49])$ surpasse significativement les deux distillations **PT-ME Interpolation Distillation** $(0,35\ [0,33,0,41]$, p-value $< 0,05)$ et **PT-ME Direct Distillation** $(0,35\ [0,31;0,36]$, p-value $< 0,01)$. Ceci peut s'expliquer, d'une part, par le fait que l'espace de paramétrisation des tâches est en trois dimensions et non deux, ce qui rend plus difficile l'apprentissage du réseau de neurones. D'autre part, le côté plus chaotique de ce problème, de par la simulation de la dynamique de contact, fait que les réseaux de neurones n'ont peut-être pas la précision requise.

5.6 Conclusion

PT-ME trouve des solutions performantes pour autant de tâches que son budget lui permet, facilitant la généralisation et la distillation de ces solutions en une politique qui propose une solution pour chaque paramétrisation de tâche. Sa généralisation surpasse significativement la méthode d'apprentissage par renforcement PPO (Schulman, Wolski et al., 2017) sur les trois problèmes étudiés.

PT-ME peut-il être utilisé avec un grand espace de paramétrisation des tâches ? PT-ME discrétise l'espace des tâches en utilisant CVT (Q. Du et al., 1999), qui fonctionne très bien en haute dimension. Cependant, l'archive requiert plus de cellules en grandes dimensions pour conserver une résolution de qualité, ce qui, comme pour MT-ME (Mouret et Maguire, 2020), ralentit la convergence vers de bonnes solutions. Pour généraliser, nous utilisons notre nouvel opérateur de variations de régression linéaire locale (avec ou sans distillation), qui pourrait aussi être ralenti par l'augmentation des dimensions (car le modèle linéaire requiert l'inversion d'une matrice). N'importe quelle technique d'apprentissage supervisé (par exemple, les réseaux de neurones) peut être utilisée comme méthode de généralisation.

PT-ME peut-il être utilisé sur de grands espaces de solutions ? Plusieurs méthodes permettent d'exploiter des informations de gradient pour permettre à MAP-Elites d'optimiser de grands espaces de solutions (par exemple, un réseau de neurones). Par exemple, exploiter la structure différentiable du problème en calculant le gradient de la fonction de *fitness* (Fontaine et Nikolaidis, 2021a) (ce qui nous éloigne des problèmes boîte noire). Ou encore, entraîner un réseau de neurones critique et exploiter sa différentiabilité (Nilsson et al., 2021) (qui reste donc applicable aux problèmes boîte noire). De telles méthodes pourraient être utilisées pour améliorer notre solution.

Chapitre 6

Discussion

Lors de cette thèse, nous nous sommes intéressés à étudier une approche en deux étapes pour apprendre des comportements à des agents artificiels. Les chapitres 3 et 4 montrent que découper l'apprentissage en une première étape de résolution et une seconde étape de généralisation permet d'apprendre un réflexe à un robot humanoïde. Le chapitre 5 montre qu'il est possible, grâce à un nouvel algorithme d'optimisation multi-tâche boîte noire, *Parametric-Task MAP-Elites*, d'obtenir une meilleure efficacité d'échantillonnage lors de la phase de résolution. *Parametric-Task MAP-Elites* résout une nouvelle tâche à chaque itération et permet d'entraîner une politique générale qui obtient de meilleures performances de généralisation qu'une méthode d'apprentissage par renforcement profond.

6.1 Comparaison avec l'apprentissage par renforcement profond

Discutons d'abord des différentes caractéristiques d'apprentissage en comparant notre méthode (résolution puis généralisation) à l'apprentissage par renforcement profond.

Passage à l'échelle. Nous avons comparé notre méthode uniquement sur des problèmes de faibles dimensions de solutions (2 pour un réflexe d'évitement de chute pour humanoïde avec un bras, 2 pour le problème jouet *Archery*, 9 pour l'ouverture de la porte avec l'humanoïde et 10 pour le problème jouet *10-DoF Arm*). En particulier, nous n'avons pas testé sur des problèmes nécessitant d'optimiser une séquence d'actions en boucle fermée. Il est possible que pour des politiques avec de petits réseaux de neurones (moins de 1000 paramètres) notre méthode obtienne de bonnes performances de la même manière que CMA-ES (D. Ha et al., 2018).

Généralité de la méthode. *Parametric-Task MAP-Elites* résout un problème d'optimisation multi-tâche qui est un problème plus général que l'apprentissage de politique. Il serait donc intéressant de voir des applications à d'autres problèmes que l'apprentissage direct de politique.

Inférence par un réseau de neurones. Notre méthode permet de conserver l'utilisation à l'inférence d'un réseau de neurones profond pour générer un comportement. À la différence d'une politique dans un algorithme d'apprentissage par renforcement profond, elle ne permet pas la sélection d'une séquence d'actions en boucle fermée. Dans le cas des problèmes de robotique, nous avons préféré l'utiliser pour sélectionner des paramètres de haut niveau d'une politique de bas niveau générée par le contrôleur du corps complet.

Utilisation des données. D'une part, les comparaisons montrent que notre méthode est plus efficace en données que PPO sur les problèmes de faible dimension testés (c'est-à-dire avec un espace de recherche de moins de 10 dimensions mais ayant des récompenses éparses). Il est par contre possible qu'une méthode d'apprentissage hors ligne, par exemple SAC (Haarnoja et al., 2018), soit plus efficace en données que PPO, qui est un apprentissage en ligne. D'autre part, notre méthode n'est pas de bout en bout et permet donc de réutiliser le jeu de données construit pour plusieurs apprentissages d'une politique générale. Ce qui permet d'être globalement plus efficace en échantillonnage en termes de l'ensemble des évaluations requises pour développer la méthode et la paramétrer.

Exploration. Notre méthode n'exploite pas explicitement de méthodes d'exploration comme une motivation intrinsèque. Cependant, l'utilisation d'une archive de solutions pour conserver une diversité de solutions (si les différentes tâches nécessitent différentes solutions) et les opérateurs de variation (en particulier SBX (Agrawal et al., 2000) et *iso-line* (Vassiliades et Mouret, 2018)) permettent d'explorer l'espace des solutions plus efficacement qu'un échantillonnage uniforme ou qu'une simple perturbation gaussienne, souvent utilisée pour les algorithmes d'apprentissage par renforcement profond. Notre nouvel opérateur de variation (Section 5.4.3) utilise une régression linéaire locale sur un voisinage d'élites. Il se place plus du côté de l'exploitation que de l'exploration et n'est pas suffisant pour résoudre l'ensemble de l'espace des tâches.

Facilité d'utilisation. Notre méthode est séparée en deux étapes distinctes. Non seulement la seconde étape d'apprentissage supervisé est plus simple à paramétrer qu'une méthode d'apprentissage par renforcement profond, mais de plus elle ne nécessite pas une nouvelle collecte et donc la phase d'ajustement des paramètres est plus rapide. De plus, il est possible d'étudier les solutions trouvées après l'étape de résolution indépendamment de la généralisation. En particulier, il est facile de savoir quelles sont les tâches réussies et les tâches échouées. Notre méthode nécessite tout de même l'ajustement d'hyperparamètres tels que la résolution de l'archive et le choix des opérateurs de variation.

Interprétation et explicabilité. Nous n'avons pas explicitement intégré de principe d'interprétabilité, mais l'étude des solutions de l'archive générée lors de la phase de résolution est plus simple que l'étude d'une politique via un réseau de neurones.

Contraintes. Nous n'avons, pour l'instant, pas considéré le respect de contraintes hormis une possible pénalité dans la fonction de *fitness*, donc comme l'apprentissage par renforcement profond. Dans le cas de la robotique, si nous considérons l'utilisation d'un contrôleur du corps complet comme une politique de bas niveau paramétrée par notre méthode, alors notre méthode prend en compte des contraintes. À l'opposée, une méthode d'apprentissage par renforcement profond qui contrôlerait le robot au niveau des articulations ne pourrait le faire qu'indirectement via la fonction de récompense.

6.2 Subdivision du problème

Pour l'instant, nous avons divisé l'apprentissage en deux phases : une première phase de résolution et une seconde phase de généralisation. Pour effectuer la première phase de résolution, nous avons proposé de subdiviser le problème général en un ensemble d'instances paramétrées que nous avons appelées "tâches". Ceci correspond à résoudre le problème de la généralisation à des variations d'un même problème (que nous pouvons aussi voir comme un

problème de robustesse ou d'adaptation). Pour résoudre ce problème, les méthodes d'apprentissage par renforcement profond apprennent une unique politique robuste qui peut s'adapter via la boucle fermée de la séquence d'états. Cette politique est entraînée le plus généralement avec de la "randomisation" de l'état initial et avec du bruit dans les actions et les états pour renforcer la robustesse.

À l'opposé, notre méthode propose d'abord de trouver des solutions particulières pour un ensemble de variations du problème et ensuite de résoudre le problème de la généralisation. Si la paramétrisation du problème permet de couvrir l'ensemble des instances possibles alors, avec un budget suffisamment élevé, l'algorithme peut espérer avoir résolu "toutes" les variations possibles du problème lors de la première étape. Dans ce cas, la seconde étape de généralisation ne pourrait correspondre qu'à une détection de la variante courante du problème pour sélectionner la solution correspondante.

Un cas différent est la généralisation à des problèmes non contenus dans l'espace de paramétrisation. Nous pouvons déjà nous demander si c'est à un expert de connaître à l'avance l'ensemble des problèmes que devra permettre de résoudre l'agent ou, au contraire, si c'est à l'agent de pouvoir être robuste ou s'adapter à des imprévus. Dans le monde réel, il semble fortement optimiste de pouvoir prévoir tous les imprévus. Une solution plus réaliste semble plutôt de permettre à l'agent d'apprendre en cours d'utilisation, par exemple en permettant à l'archive de solutions de collecter de nouvelles solutions pour de nouvelles tâches.

Un aspect de l'instanciation du problème en un ensemble de tâches qui n'a pas encore été discuté est l'optimisation de la répartition des efforts pour résoudre les différentes tâches. Notre méthode actuelle sélectionne aléatoirement la tâche sur laquelle va être évaluée la solution candidate, soit uniformément, soit indirectement en biaisant à l'aide d'un tournoi vers une tâche proche de la solution d'un parent élite. Une raison pour laquelle il peut être intéressant de choisir les tâches sur lesquelles accorder plus de budgets d'évaluations est qu'il peut y avoir une disparité de difficulté à résoudre les différentes tâches. Continuer de chercher à optimiser une solution d'une tâche facile déjà résolue gaspille donc une partie du budget. Une difficulté est de pouvoir savoir si une tâche est résolue lorsque la *fitness* maximale n'est pas connue. De plus, biaiser l'échantillonnage vers des tâches non encore résolues pose le problème des tâches impossibles sur lesquelles il ne faut pas optimalement gaspiller des évaluations. Le principe d'élitisme mou (Fontaine et Nikolaidis, 2023) permet par exemple de réguler le biais de résolution entre explorations de tâches non résolues et affinage des solutions des tâches déjà résolues. Des méthodes de curiosité intrinsèque (Oudeyer et al., 2007 ; Aubret et al., 2019) pourraient aussi être utiles avec, par exemple, les mesures de progrès qui permettent de biaiser vers des tâches ni trop faciles ni trop difficiles. Nous pouvons aussi de nouveau mettre en lumière la similarité entre notre optimisation multi-tâche avec une archive et les méthodes d'exploration dirigée par des buts intrinsèquement motivés (Baranes et al., 2013) qui construisent un modèle inverse entre des politiques et des comportements. Ces méthodes sélectionnent la tâche à partir d'une mesure d'intérêt puis la résolvent sans directement exploiter la composante multi-tâche du problème global. À L'opposé, nous utilisons des méthodes multi-tâches pour améliorer l'efficacité d'échantillonnage, mais en sélectionnant aléatoirement une tâche à chaque itération. Explorer la fusion des deux méthodes pourrait proposer des résultats intéressants.

Pour la phase de résolution, nous avons, pour l'instant, divisé le problème en différentes instances de même niveau. Une autre division possible est de diviser de manière séquentielle. Diviser séquentiellement permet de découper un problème en plusieurs tâches individuellement plus simples à résoudre. Par exemple, pour ouvrir une porte, apprendre à attraper la poignée puis apprendre à tirer la poignée. Par rapport à notre formulation actuelle qui formalise ces

deux tâches en une seule, subdiviser en deux sous-tâches permet d'optimiser chacune indépendamment. Pour la formulation actuelle, le robot ne peut pas apprendre à tirer la poignée tant qu'il n'a pas appris à la saisir. De plus, une fois que la saisie de la poignée est maîtrisée, pour optimiser le tirage de la poignée, il peut être néfaste de continuer à appliquer des variations sur les paramètres qui régulent la saisie. En séparant séquentiellement de manière explicite, l'algorithme peut apprendre une première sous-tâche puis la fixer pour apprendre la sous-tâche suivante.

Pratiquer un découpage séquentiel explicite nécessite d'avoir un expert qui sait comment subdiviser le problème. Une formulation à l'aide d'une politique séquentielle, comme pour l'apprentissage par renforcement profond, est plus généraliste. Apprendre deux politiques comme Go-Explore (Ecoffet et al., 2021 ; Norman et al., 2023) permet, en un sens, d'obtenir les avantages des deux méthodes. D'une part, la construction d'une archive d'états atteignables et l'apprentissage d'une politique qui permet de reproduire ces comportements pour retourner à l'un de ces états permet de fixer des sous-tâches résolues. D'autre part, l'utilisation d'une méthode d'exploration qui part donc des comportements appris pour étendre l'espace des comportements permet d'agrandir l'archive.

Une autre subdivision du problème que l'instanciation ou le séquençage est la hiérarchisation. Hiérarchiser permet d'abord d'apprendre des primitives qui peuvent être recombinées pour créer un comportement plus complexe, de la même manière qu'un enfant apprend d'abord à maîtriser son corps pour ensuite maîtriser son environnement (Gopnik et al., 1999). Hiérarchiser permet d'obtenir une complexification exponentielle tout en conservant des composants individuellement plus simples. L'archive peut, par exemple, être utilisée comme un ensemble de primitives utilisées par une politique de plus haut niveau (Duarte et al., 2018). Mais l'archive en elle-même peut être hiérarchique pour permettre l'apprentissage de comportements de plus en plus complexes (Allard et al., 2022).

6.3 Généralisation

Qualité-diversité multi-tâche. Les chapitres 3 et 4 s'intéressent au problème de la qualité-diversité multi-tâche. Le chapitre 5 propose un nouvel algorithme d'optimisation multi-tâche. Nous n'avons pas appliqué PT-ME aux problèmes d'apprentissage de réflexes, car nous voulions d'abord pallier une faiblesse des algorithmes d'optimisation multi-tâche avant de nous pencher sur le problème plus complexe de qualité-diversité multi-tâche. Une extension directe est donc d'étendre PT-ME au problème de qualité-diversité de tâches paramétriques. Ceci pourrait se faire en ajoutant, comme pour *Multi-Task Multi-Behavior MAP-Elites*, une dimension de comportement dans chaque cellule de l'archive.

Obtenir une diversité comportementale a plusieurs utilités. Premièrement, elle permet de guider l'exploration des solutions hors des minima locaux (Lehman et al., 2008). Deuxièmement, le classifieur entraîné grâce à une diversité comportementale permet d'obtenir "gratuitement" un comportement robuste, c'est-à-dire dont les comportements proches sont aussi réussis (Chapitre 3). Troisièmement, connaître une diversité comportementale qui résout la tâche permet, au moment de l'inférence, de pouvoir s'adapter à un imprévu, par exemple, une surface quelconque et non plus parfaite (Chapitre 4).

Alternatives à la distillation directe des solutions. Nous avons utilisé plusieurs méthodes de généralisation : un réseau de neurones de classification qui s'entraîne sur une diversité de comportements, un réseau de neurones de régression qui apprend directement les élites de l'archive et une régression locale linéaire des élites de l'archive.

La classification présente deux avantages par rapport à la régression. D'une part, le problème est plus facile (Torgo et al., 1996 ; Salman et al., 2012 ; Farebrother et al., 2024). D'autre part, la classification permet de conserver lors de l'inférence une diversité de solutions (Chapitre 3). Un inconvénient est la nécessité de devoir définir un critère binaire de réussite qui est encore plus arbitraire qu'une mesure de qualité continue.

À l'opposé, la régression a l'avantage de proposer directement une solution, ce qui simplifie l'inférence. Une régression via un réseau de neurones peut également conserver une diversité de solutions en étant conditionnée par un état cible (Jegorova et al., 2020) ou un comportement cible (Faldor et al., 2023). De plus, au lieu d'être apprise par simple régression, la politique peut être entraînée à l'aide d'un algorithme d'apprentissage par imitation, ce qui permet de la renforcer (Ecoffet et al., 2021). Selon la méthode utilisée, cela a l'inconvénient de nécessiter des évaluations supplémentaires, mais reste tout de même plus rapide que de trouver directement des politiques robustes par apprentissage par renforcement profond.

L'interpolation sous forme de régression linéaire locale (Section 5.4.5.1) montre que l'archive en elle-même est déjà un outil puissant qui ne nécessite pas davantage d'apprentissage, du moins pour les problèmes étudiés qui sont de faible dimensionnalité. Elle peut aussi être utilisée comme une bonne méthode de référence pendant une phase de débogage pour étudier l'apport d'une méthode d'apprentissage supplémentaire pour la généralisation.

Une approche totalement différente d'utilisation du jeu de données de comportements est l'apprentissage du sous-espace des comportements performants (Gaier et al., 2020), dans le but d'utiliser ce sous-espace lors de l'inférence comme espace de recherche pour une méthode d'optimisation quelconque. Cela a pour inconvénient de ne pas être une simple inférence rapide, étant donné qu'il faut de nouveau optimiser dans ce sous-espace pour obtenir une solution. Mais cela pourrait permettre d'obtenir de l'adaptation rapide, par exemple avec une méthode d'essai-erreur qui exploite ce sous-espace pour trouver rapidement des solutions diverses et de bonne qualité.

6.4 L'apprentissage

L'apprentissage direct de réseaux de neurones profonds. Une alternative à notre méthode et à l'apprentissage par renforcement profond est l'optimisation directe de réseaux de neurones profonds avec des méthodes de la robotique évolutionniste. Des méthodes telles que PGA-ME (Nilsson et al., 2021) et MEGA (Fontaine et Nikolaidis, 2021b) ont montré la capacité des algorithmes de la famille MAP-Elites à optimiser directement des réseaux de neurones profonds tout en profitant des principes de diversité comportementale. Optimiser directement une politique séquentielle en boucle fermée permet de représenter un espace de comportement beaucoup plus grand. Cela nécessite donc moins de connaissances d'un expert, mais plus d'échantillons pour découvrir des comportements performants.

L'apprentissage continu. Pour que des agents artificiels puissent être déployés dans le monde réel de façon pérenne, il est nécessaire qu'ils soient dotés d'une capacité à s'adapter et à apprendre tout au long de leur utilisation (Thrun et Mitchell, 1995). Les réseaux de neurones profonds sont assez bons pour compresser de l'information, mais souffrent de pertes de connaissances du fait de leur taille limitée (Toneva et al., 2018). L'apprentissage par renforcement profond ne semble pas être la méthode générale pour cela, bien qu'il puisse être l'un des composants permettant l'acquisition ou le raffinement d'un comportement pour une tâche particulière. L'utilisation d'une archive pour organiser les comportements acquis permet d'éviter des problèmes d'oubli (en supposant que l'archive peut croître de façon arbitraire).

Pour être opérante, une telle formulation nécessite plusieurs composants. Un premier composant nécessaire est une représentation des tâches qui permet de les comparer pour trouver les tâches similaires. Il est difficile de trouver une représentation générale identique pour l'ensemble des tâches. Il faut donc recourir à des techniques multi-tâches explicites pour permettre un échange d'information entre des représentations différentes (Feng, L. Zhou et al., 2018). Ceci peut aussi être contenu dans une organisation hiérarchique de l'archive, par exemple avec des primitives de bas niveau de contrôle utilisées par un niveau supérieur pour réaliser différentes tâches.

Un autre composant nécessaire est une méthode de gestion de l'archive pour reconnaître si la tâche courante est déjà résolue ou nécessite l'apprentissage d'un nouveau comportement. Ceci implique l'utilisation d'une méthode d'apprentissage d'un nouveau comportement en exploitant, si possible, les comportements déjà appris.

Les grands modèles de langage (LLMs). L'une des dernières grandes avancées de l'apprentissage ces dernières années sont les grands modèles de langage (W. X. Zhao et al., 2023). Ils apportent deux résultats impressionnants : la compression du savoir de l'humanité (avec perte et risque d'hallucination) et la possibilité de communiquer naturellement avec un agent artificiel. Ces résultats s'illustrent aussi pour la robotique (Zeng et al., 2023). Par exemple, OpenAI a récemment montré [12] leur capacité à permettre à un robot humanoïde de décrire son environnement, de comprendre les tâches qui lui sont demandées, d'activer le comportement correspondant puis d'évaluer sa réalisation.

Nous parlions un peu plus haut de la difficulté de trouver une représentation commune à l'ensemble des tâches possibles que peut rencontrer un agent durant son utilisation. Le langage naturel peut être un bon candidat pour une telle représentation. Non seulement il permet de décrire l'ensemble des tâches possibles, mais il permet aussi une compréhension rapide par un humain. La connaissance d'un savoir général de l'humanité peut aussi être utilisée pour permettre à l'agent de reconnaître et comparer les tâches. Par exemple, un agent peut se rendre compte qu'il va devoir ouvrir une porte puis regarder dans son archive s'il sait ouvrir une porte. L'acte d'ouvrir une porte peut aussi être décomposé naturellement en une séquence de sous-tâches, chacune résolue par une primitive.

Pourquoi utiliser de l'apprentissage ? Nous pouvons distinguer deux intentions d'utilisation de l'apprentissage pour qu'un agent artificiel puisse résoudre une tâche. D'un côté, il y a la volonté d'utiliser une méthode efficace pour doter l'agent d'un comportement avec pour unique but que l'agent puisse résoudre la tâche. De l'autre côté, il y a la volonté de reproduire et d'étudier les mécanismes biologiques d'apprentissage. Si le but est uniquement de doter un agent artificiel d'un comportement, il ne faut pas ignorer l'ensemble des outils à notre disposition, bien qu'ils ne proviennent pas d'une inspiration du vivant. Ainsi, avant de choisir d'utiliser une méthode d'apprentissage par renforcement profond, il faut se poser la question de si c'est la meilleure solution. C'est peut-être le cas, par exemple, pour de nouvelles morphologies d'agent où aucun expert n'a de connaissances sur ses capacités et où l'apprentissage par interaction unique avec l'environnement est la seule solution.

Même en restant dans une volonté d'imiter le vivant, un humain n'apprend pas purement à l'aide d'interactions avec son environnement (Vygotsky et al., 1978). Même sans prendre en compte d'éventuels biais génétiques, un être vivant n'apprend pas chaque nouveau comportement en partant de zéro, mais à la suite d'une longue série d'expériences vécues. La formulation multi-tâche permet de modéliser l'un de ces aspects en permettant de factoriser l'apprentissage de plusieurs comportements et en amorçant l'apprentissage d'un nouveau

12. https://youtu.be/Sq1QZB5baNw

comportement à l'aide d'expériences passées. La méthode actuelle suppose une seule phase de résolution des différentes tâches. Il faudrait aller plus loin en permettant l'ajout de nouvelles tâches dans le temps.

L'éducation fait aussi partie des méthodes d'apprentissage des individus d'une espèce (Bransford et al., 2000), et pas uniquement de l'espèce humaine (Hoppitt et al., 2008 ; Thornton et al., 2008). Pour les agents artificiels, elle peut prendre la forme d'un apprentissage par démonstration avec l'aide d'un expert humain, ce qui permet d'accélérer l'apprentissage tout en permettant par la suite d'utiliser des méthodes d'apprentissage autonome pour affiner le comportement. L'éducation peut aussi se formaliser comme un apprentissage multi-agent, où chaque agent partage avec ses congénères des informations.

En suivant les principes de la robotique évolutionniste, certains auteurs (G. E. Hinton et al., 1987 ; Urzelai et al., 2001 ; Soltoggio et al., 2007 ; Risi et al., 2010) envisagent même un apprentissage mêlant plusieurs formes d'acquisition de connaissances, par exemple par voie génétique d'un parent à un enfant, par échange entre les individus d'une même génération et par interaction autonome avec l'environnement. Cette interférence de différents modes d'apprentissage forme un système dynamique complexe à étudier et à paramétrer, mais avec un fort potentiel.

Chapitre 7

Conclusion

Cette thèse étudie une alternative à l'apprentissage par renforcement profond en divisant l'apprentissage en une étape de résolution et une étape de généralisation. Séparer en deux étapes simplifie globalement l'utilisation en permettant : de pouvoir utiliser des méthodes dédiées à chaque étape, de ne pas nécessiter une nouvelle collecte de données à chaque apprentissage et de pouvoir étudier chaque étape séparément.

L'étape de résolution consiste à séparer le problème global en un ensemble de sous-problèmes plus faciles à résoudre tout en permettant l'utilisation d'algorithme d'optimisation multi-tâche pour exploiter la similarité entre les différentes tâches. Le chapitre 3 utilise une simple recherche en grille sur chaque tâche séparément (ce qui est efficace parce que l'espace de recherche est de dimension deux). Le chapitre 4 propose une nouvelle méthode de qualité-diversité multi-tâche, *Multi-Task Multi-Behavior MAP-Elites*, qui exploite la diversité comportementale et la similarité entre les tâches pour être plus efficace en échantillonnage qu'une recherche par grille. Le chapitre 5 propose une généralisation de notre méthode au problème d'optimisation multi-tâche, *Parametric-Task MAP-Elites*, qui résout une nouvelle tâche à chaque itération. L'optimisation est plus générale que le problème d'apprentissage de politique mais peut s'y appliquer sous la forme d'une recherche directe de politique.

L'étape de généralisation consiste à utiliser de l'apprentissage supervisé sur le jeu de données créé à la première étape. Les chapitres 3 et 4 utilisent un classifieur pour conserver une diversité de solutions, ce qui permet d'être plus robuste et de pouvoir s'adapter. Le chapitre 5 propose de distiller les solutions sous forme d'une régression qui permet de proposer une solution de qualité pour l'ensemble de l'espace des tâches.

Cette thèse montre que cet apprentissage de politique en deux étapes est viable et même plus efficace que la méthode d'apprentissage par renforcement profond PPO (Schulman, Wolski et al., 2017) sur un ensemble de problèmes testés de faible dimension (moins de 10) mais avec des récompenses éparses. Les chapitres 3 et 4 montrent que la méthode permet d'apprendre un réflexe d'évitement de chute à un robot humanoïde suffisamment robuste pour s'adapter au robot réel. Le chapitre 5 montre sur deux problèmes jouets et un problème plus réaliste de robotique que la méthode est performante lorsque la fonction de *fitness* est éparse, un exemple typique où la mauvaise efficacité d'échantillonnage et d'exploration de l'apprentissage par renforcement profond se fait ressentir.

Notre méthode a plusieurs limitations. Elle n'a pas encore été testée pour des problèmes de grandes dimensions et pour des politiques en boucle fermée. Elle n'exploite aucune méthode directe d'exploration. Enfin, elle ne propose pas de mécanismes pour l'interprétabilité des comportements trouvés ou pour le respect des contraintes.

L'étape suivante consiste à adapter *Parametric-Task MAP-Elites* pour le problème de qualité-diversité multi-tâche. Un travail futur sera d'exploiter les opérateurs de variations récents et plus performants qui exploitent les stratégies d'évolution (Fontaine et Nikolaidis, 2023) ou des gradients de politique (Nilsson et al., 2021 ; Fontaine et Nikolaidis, 2021a). L'utilisation d'une motivation intrinsèque telle que le progrès d'apprentissage (Baranes et al., 2013) pourrait permettre d'optimiser la répartition des évaluations entre les différentes tâches.

Une première perspective à plus long terme est ce que les grands modèles de langues peuvent apporter, par exemple pour obtenir une représentation généraliste des tâches et comportements. Une seconde perspective est d'étudier d'étudier d'autres formes d'archives (par exemple séquentielles, hiérarchiques ou croissantes) pour permettre l'apprentissage de comportements plus complexes et de façon continue dans le temps.

Bibliographie

Aasi, J., J. Abadie, B. Abbott, R. Abbott, T. Abbott, M. Abernathy, T. Accadia, F. Acernese, C. Adams, T. Adams et al. (2013). « Einstein@ Home all-sky search for periodic gravitational waves in LIGO S5 data ». In : *Physical Review D* 87.4, p. 042001.

Agarap, A. F. (2018). « Deep learning using rectified linear units (relu) ». In : *arXiv preprint :1803.08375*.

Agrawal, R., K. Deb et R. Agrawal (juin 2000). « Simulated Binary Crossover for Continuous Search Space ». In : *Complex Systems* 9.

Alarie, S., C. Audet, V. Garnier, S. Le Digabel et L.-A. Leclaire (2013). « Snow water equivalent estimation using blackbox optimization ». In : *Pac J Optim* 9.1, p. 1-21.

Alarie, S., C. Audet, A. E. Gheribi, M. Kokkolaras et S. Le Digabel (2021). « Two decades of blackbox optimization applications ». In : *EURO Journal on Computational Optimization* 9, p. 100011. ISSN : 2192-4406. DOI : https://doi.org/10.1016/j.ejco.2021.100011.

Allard, M., S. C. Smith, K. Chatzilygeroudis et A. Cully (2022). « Hierarchical quality-diversity for online damage recovery ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 58-67.

Amodei, D. et D. Hernandez (2018). *AI and compute*. URL : https://openai.com/research/ai-and-compute (visité le 01/03/2024).

Anne, T., E. Dalin, I. Bergonzani, S. Ivaldi et J.-B. Mouret (2022). « First Do Not Fall : Learning to Exploit a Wall With a Damaged Humanoid Robot ». In : *IEEE Robotics Autom. Lett.* 7.4, p. 9028-9035. DOI : 10.1109/LRA.2022.3188884.

Anne, T. et J.-B. Mouret (2023). « Multi-Task Multi-Behavior MAP-Elites ». In : *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023*. Sous la dir. de S. Silva et Paquete. ACM, p. 111-114. DOI : 10.1145/3583133.3590730.

— (2024). « Parametric-Task MAP-Elites ». In : *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2024, Companion Volume, Melbourne, Australia, July 14-18, 2024*. ACM. DOI : 10.1145/3638529.3653993.

Anne, T., J. Wilkinson et Z. Li (2021). « Meta-Learning for Fast Adaptive Locomotion with Uncertainties in Environments and Robot Dynamics ». In : *IEEE/RSJ IROS*, p. 4568-4575. DOI : 10.1109/IROS51168.2021.9635840.

Arulkumaran, K., M. P. Deisenroth, M. Brundage et A. A. Bharath (2017). « Deep reinforcement learning : A brief survey ». In : *IEEE Signal Processing Magazine* 34.6, p. 26-38.

Atkeson, C. et al. (2018). « What Happened at the DARPA Robotics Challenge Finals ». In : *The DARPA Robotics Challenge Finals : Humanoid Robots To The Rescue*, p. 667-684. ISBN : 978-3-319-74665-4. DOI : 10.1007/978-3-319-74666-1_17.

Aubret, A., L. Matignon et S. Hassas (2019). « A survey on intrinsic motivation in reinforcement learning ». In : *CoRR* abs/1908.06976. arXiv : 1908.06976.

Audet, C., V. Béchard et J. Chaouki (2008). « Spent potliner treatment process optimization using a MADS algorithm ». In : *Optimization and Engineering* 9, p. 143-160.

Auer, P., N. Cesa-Bianchi et P. Fischer (mai 2002). « Finite-time Analysis of the Multiarmed Bandit Problem ». In : *Machine Learning* 47, p. 235-256. DOI : 10.1023/A:1013689704352.

Back, T., U. Hammel et H.-P. Schwefel (1997). « Evolutionary computation : comments on the history and current state ». In : *IEEE Transactions on Evolutionary Computation* 1.1, p. 3-17. DOI : 10.1109/4235.585888.

Bali, K. K., A. Gupta, Y.-S. Ong et P. S. Tan (2021). « Cognizant Multitasking in Multiobjective Multifactorial Evolution : MO-MFEA-II ». In : *IEEE Transactions on Cybernetics* 51.4, p. 1784-1796. DOI : 10.1109/TCYB.2020.2981733.

Bali, K. K., Y.-S. Ong, A. Gupta et P. S. Tan (2020). « Multifactorial Evolutionary Algorithm With Online Transfer Parameter Estimation : MFEA-II ». In : *IEEE Transactions on Evolutionary Computation* 24.1, p. 69-83. DOI : 10.1109/TEVC.2019.2906927.

Baranes, A. et P.-Y. Oudeyer (2013). « Active learning of inverse models with intrinsically motivated goal exploration in robots ». In : *Robotics and Autonomous Systems* 61.1, p. 49-73.

Barnett, S. (1968). « A simple class of parametric linear programming problems ». In : *Operations Research* 16.6, p. 1160-1165.

Barto, A. G., R. S. Sutton et C. W. Anderson (1983). « Neuronlike adaptive elements that can solve difficult learning control problems ». In : *IEEE transactions on systems, man, and cybernetics* 5, p. 834-846.

Baydin, A. G., B. A. Pearlmutter, A. A. Radul et J. M. Siskind (2018). « Automatic differentiation in machine learning : a survey ». In : *Journal of machine learning research* 18.153, p. 1-43.

Bellemare, M. G., Y. Naddaf, J. Veness et M. Bowling (2013). « The arcade learning environment : An evaluation platform for general agents ». In : *Journal of Artificial Intelligence Research* 47, p. 253-279.

Bellman, R. (1966). « Dynamic programming ». In : *Science* 153.3731, p. 34-37.

Berry, D. A. et B. Fristedt (1985). « Bandit problems : sequential allocation of experiments (Monographs on statistics and applied probability) ». In : *London : Chapman and Hall* 5.71-87, p. 7-7.

Betz, T., H. Fujiishi et T. Kobayashi (2021). « Behavioral cloning from observation with bidirectional dynamics model ». In : *2021 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, p. 184-189.

Beyer, H.-G. et H.-P. Schwefel (2002). « Evolution strategies–a comprehensive introduction ». In : *Natural computing* 1, p. 3-52.

Bongard, J. (2013). « Evolutionary robotics ». In : *Communications of the ACM* 56.8, p. 74-83.

Bouyarmane, K. et A. Kheddar (2017). « On weight-prioritized multitask control of humanoid robots ». In : *IEEE Transactions on Automatic Control* 63.6, p. 1632-1647.

Bransford, J. D., A. L. Brown, R. R. Cocking et al. (2000). *How people learn*. T. 11. Washington, DC : National academy press.

Braylan, A., M. Hollenbeck, E. Meyerson et R. Miikkulainen (2015). « Frame skip is a powerful parameter for learning to play atari ». In : *Workshops at the twenty-ninth AAAI conference on artificial intelligence*.

Brochu, E., V. M. Cora et N. de Freitas (2010). « A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning ». In : *CoRR* abs/1012.2599. arXiv : 1012.2599.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang et W. Zaremba (2016). « Openai gym ». In : *arXiv preprint :1606.01540*.

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al. (2020). « Language models are few-shot learners ». In : *Advances in neural information processing systems* 33, p. 1877-1901.

Bruin, T. de, J. Kober, K. Tuyls et R. Babuška (2020). « Fine-tuning deep RL with gradient-free optimization ». In : *IFAC-PapersOnLine* 53.2, p. 8049-8056.

Chatzilygeroudis, K. et J.-B. Mouret (2018). « Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics ». In : *IEEE ICRA*.

Chatzilygeroudis, K., R. Rama, R. Kaushik, D. Goepp, V. Vassiliades et J.-B. Mouret (2017). « Black-box data-efficient policy search for robotics ». In : *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, p. 51-58.

Chatzilygeroudis, K., V. Vassiliades, F. Stulp, S. Calinon et J.-B. Mouret (2019). « A survey on policy search algorithms for learning robot controllers in a handful of trials ». In : *IEEE Transactions on Robotics* 36.2, p. 328-347.

Chebotar, Y., K. Hausman, M. Zhang, G. Sukhatme, S. Schaal et S. Levine (2017). « Combining model-based and model-free updates for trajectory-centric reinforcement learning ». In : *International conference on machine learning*. PMLR, p. 703-711.

Chebotar, Y., M. Kalakrishnan, A. Yahya, A. Li, S. Schaal et S. Levine (2017). « Path integral guided policy search ». In : *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, p. 3381-3388.

Chen, D., Y. Wang et W. Gao (2020). « Combining a gradient-based method and an evolution strategy for multi-objective reinforcement learning ». In : *Applied Intelligence* 50, p. 3301-3317.

Chen, W., C. Gao et W. Jing (2023). « Proximal policy optimization guidance algorithm for intercepting near-space maneuvering targets ». In : *Aerospace Science and Technology* 132, p. 108031.

Chi, C., S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel et S. Song (2023). « Diffusion policy : Visuomotor policy learning via action diffusion ». In : *arXiv preprint :2303.04137*.

Chitta, S., B. Cohen et M. Likhachev (2010). « Planning for autonomous door opening with a mobile manipulator ». In : *2010 IEEE International Conference on Robotics and Automation*, p. 1799-1806. DOI : 10.1109/ROBOT.2010.5509475.

Chong, E. K. et S. H. Żak (2013). *An introduction to optimization*. T. 75. John Wiley & Sons.

Chua, K., R. Calandra, R. McAllister et S. Levine (2018). « Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi et R. Garnett. T. 31. Curran Associates, Inc.

Cideron, G., T. Pierrot, N. Perrin, K. Beguir et O. Sigaud (2020). « Qd-rl : Efficient mixing of quality and diversity in reinforcement learning ». In : *arXiv preprint :2006.08505*, p. 36.

Colas, C., V. Madhavan, J. Huizinga et J. Clune (2020). « Scaling map-elites to deep neuroevolution ». In : *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, p. 67-75.

Conti, E., V. Madhavan, F. Petroski Such, J. Lehman, K. O. Stanley et J. Clune (2018). « Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi et R. Garnett. T. 31. Curran Associates, Inc.

Cui, D. et al. (2021). « Human inspired fall arrest strategy for humanoid robots based on stiffness ellipsoid optimisation ». In : *Bioinspiration & biomimetics* 16. DOI : 10.1088/1748-3190/ac1ab9.

Cully, A. (2021). « Multi-emitter map-elites : improving quality, diversity and data efficiency with heterogeneous sets of emitters ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 84-92.

Cully, A., J. Clune, D. Tarapore et J.-B. Mouret (2015). « Robots that can adapt like animals ». In : *Nature* 521.7553, p. 503-507. DOI : 10.1038/nature14422.

Cully, A. et Y. Demiris (2018). « Hierarchical behavioral repertoires with unsupervised descriptors ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 69-76.

Cully, A. et J.-B. Mouret (2016). « Evolving a behavioral repertoire for a walking robot ». In : *Evolutionary computation* 24.1, p. 59-88.

Cunningham, P., M. Cord et S. J. Delany (2008). « Supervised learning ». In : *Machine learning techniques for multimedia : case studies on organization and retrieval*. Springer, p. 21-49.

Cybenko, G. (1989). « Approximation by superpositions of a sigmoidal function ». In : *Mathematics of control, signals and systems* 2.4, p. 303-314.

Dai, W., Z. Wang et K. Xue (mars 2022). « System-in-package design using multi-task memetic learning and optimization ». In : *Memetic Computing* 14, p. 1-15. DOI : 10.1007/s12293-021-00346-5.

Dalin, E. et al. (2021). « Whole-body teleoperation of the Talos humanoid robot : preliminary results ». In : *ICRA 2021 Workshop on Teleoperation*.

Dantec, E. L. et al. (2021). « Whole Body Model Predictive Control with a Memory of Motion : Experiments on a Torque-Controlled Talos ». In : *IEEE ICRA*. DOI : 10.1109/ICRA48506.2021.9560742.

Darwin, C. (1859). « On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life ». In : *London : Murray*.

De Boer, P.-T., D. P. Kroese, S. Mannor et R. Y. Rubinstein (2005). « A tutorial on the cross-entropy method ». In : *Annals of operations research* 134, p. 19-67.

Deisenroth, M. P., G. Neumann, J. Peters et al. (2013). « A survey on policy search for robotics ». In : *Foundations and Trends® in Robotics* 2.1–2, p. 1-142.

Deisenroth, M. P. et C. Rasmussen (2011). « PILCO : A Model-Based and Data-Efficient Approach to Policy Search ». In : *ICML 2011*, p. 465-472.

Delaunay, B. N. (1928). « Sur la sphere vide ». In : *Proceedings of the Mathematics, Toronto*. 11-16 August 1924. Toronto, p. 695-700.

Doncieux, S., N. Bredeche, J.-B. Mouret et A. E. Eiben (2015a). « Evolutionary robotics : what, why, and where to ». In : *Frontiers in Robotics and AI* 2, p. 4.

— (2015b). « Evolutionary robotics : what, why, and where to ». In : *Frontiers in Robotics and AI* 2, p. 4.

Doncieux, S., A. Laflaquière et A. Coninx (2019). « Novelty search : a theoretical perspective ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 99-106.

Doncieux, S. et J.-B. Mouret (2014). « Beyond black-box optimization : a review of selective pressures for evolutionary robotics ». In : *Evolutionary Intelligence* 7, p. 71-93.

Doncieux, S., J.-B. Mouret, N. Bredeche et V. Padois (2011). « Evolutionary robotics : Exploring new horizons ». In : *New horizons in evolutionary robotics : Extended contributions from the 2009 EvoDeRob Workshop*. Springer, p. 3-25.

Dorfman, R. (1969). « An Economic Interpretation of Optimal Control Theory ». In : *The American Economic Review* 59.5, p. 817-831. ISSN : 00028282.

Du, J., J. Fu et C. Li (2021). « Guided policy search methods : A review ». In : *Journal of Physics : Conference Series*. T. 1748. 2. IOP Publishing, p. 022039.

Du, Q., V. Faber et M. Gunzburger (1999). « Centroidal Voronoi Tessellations : Applications and Algorithms ». In : *SIAM Review* 41.4, p. 637-676. DOI : 10.1137/S0036144599352836.

Dua, P., K. Kouramas, V. Dua et E. Pistikopoulos (2008). « MPC on a chip—Recent advances on the application of multi-parametric model-based control ». In : *Computers and Chemical Engineering* 32.4. Festschrift devoted to Rex Reklaitis on his 65th Birthday, p. 754-765. ISSN : 0098-1354. DOI : https://doi.org/10.1016/j.compchemeng.2007.03.008.

Duan, Y., J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever et P. Abbeel (2016). « Rl2 : Fast reinforcement learning via slow reinforcement learning ». In : *arXiv preprint 1611.02779*.

Duarte, M., J. Gomes, S. M. Oliveira et A. L. Christensen (2016). « EvoRBC : Evolutionary Repertoire-based Control for Robots with Arbitrary Locomotion Complexity ». In : *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO '16. Denver, Colorado, USA : Association for Computing Machinery, p. 93-100. ISBN : 9781450342063. DOI : 10.1145/2908812.2908855.

— (2018). « Evolution of Repertoire-Based Control for Robots With Complex Locomotor Systems ». In : *IEEE Transactions on Evolutionary Computation* 22.2, p. 314-328. DOI : 10.1109/TEVC.2017.2722101.

Earle, S., J. Snider, M. Fontaine, S. Nikolaidis et J. Togelius (2022). « Illuminating diverse neural cellular automata for level generation ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 68-76.

Ecoffet, A., J. Huizinga, J. Lehman, K. O. Stanley et J. Clune (2021). « First return, then explore ». In : *Nature* 590.7847, p. 580-586.

Eiben, A. E. et J. E. Smith (2003). *Introduction to evolutionary computing*. Springer.

Faldor, M., F. Chalumeau, M. Flageat et A. Cully (2023). « MAP-Elites with Descriptor-Conditioned Gradients and Archive Distillation into a Single Policy ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Lisbon, Portugal : Association for Computing Machinery, p. 138-146. ISBN : 9798400701191. DOI : 10.1145/3583131.3590503.

Farebrother, J., J. Orbay, Q. Vuong, A. A. Taiga, Y. Chebotar, T. Xiao, A. Irpan, S. Levine, P. S. Castro, A. Faust et al. (2024). « Stop Regressing : Training Value Functions via Classification for Scalable Deep RL ». In : *arXiv preprint :2403.03950*.

Feng, L., Y. Huang, L. Zhou, J. Zhong, A. Gupta, K. Tang et K. C. Tan (2021). « Explicit Evolutionary Multitasking for Combinatorial Optimization : A Case Study on Capacitated Vehicle Routing Problem ». In : *IEEE Transactions on Cybernetics* 51.6, p. 3143-3156. DOI : 10.1109/TCYB.2019.2962865.

Feng, L., L. Zhou, J. Zhong, A. Gupta, Y.-S. Ong, K.-C. Tan et A. K. Qin (2018). « Evolutionary multitasking via explicit autoencoding ». In : *IEEE transactions on cybernetics* 49.9, p. 3457-3470.

Fiacco, A. V. (déc. 1976). « Sensitivity Analysis for Nonlinear Programming Using Penalty Methods ». In : *Math. Program.* 10.1, p. 287-311. ISSN : 0025-5610. DOI : 10.1007/BF01580677.

Finn, C., P. Abbeel et S. Levine (2017). « Model-agnostic meta-learning for fast adaptation of deep networks ». In : *International conference on machine learning*. PMLR, p. 1126-1135.

Fioravanzo, S. et G. Iacca (2021). « Map-elites for constrained optimization ». In : *Constraint Handling in Metaheuristics and Applications*, p. 151-173.

Fischler, M. et R. Bolles (1981). « Random Sample Consensus : A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography ». In : *Communications of the ACM* 24.6.

Flageat, M., F. Chalumeau et A. Cully (2023). « Empirical analysis of PGA-MAP-Elites for Neuroevolution in Uncertain Domains ». In : *ACM Transactions on Evolutionary Learning* 3.1, p. 1-32.

Fletcher, R. (1987). *Practical Methods of Optimization*. Second. New York, NY, USA : John Wiley & Sons.

Florence, P., C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch et J. Tompson (août 2022). « Implicit Behavioral Cloning ». In : *Proceedings of the 5th Conference on Robot Learning*. Sous la dir. d'A. Faust, D. Hsu et G. Neumann. T. 164. Proceedings of Machine Learning Research. PMLR, p. 158-168.

Foehn, P., A. Romero et D. Scaramuzza (2021). « Time-optimal planning for quadrotor waypoint flight ». In : *Science robotics* 6.56, eabh1221.

Fogel, L. J., A. J. Owens et M. J. Walsh (1966). « Artificial intelligence through simulated evolution. » In.

Fontaine, M., Y.-C. Hsu, Y. Zhang, B. Tjanaka et S. Nikolaidis (2021). « On the importance of environments in human-robot coordination ». In : *arXiv preprint :2106.10853*.

Fontaine, M., R. Liu, A. Khalifa, J. Modi, J. Togelius, A. K. Hoover et S. Nikolaidis (2021). « Illuminating mario scenes in the latent space of a generative adversarial network ». In : *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 35. 7, p. 5922-5930.

Fontaine, M. et S. Nikolaidis (2020). « A quality diversity approach to automatically generating human-robot interaction scenarios in shared autonomy ». In : *arXiv preprint :2012.04283*.

— (2021a). « Differentiable Quality Diversity ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang et J. W. Vaughan. T. 34. Curran Associates, Inc., p. 10040-10052.

— (2021b). « Differentiable quality diversity ». In : *Advances in Neural Information Processing Systems* 34, p. 10040-10052.

— (2023). « Covariance Matrix Adaptation MAP-Annealing ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Lisbon, Portugal : Association for Computing Machinery, p. 456-465. ISBN : 9798400701191. DOI : 10.1145/3583131.3590389.

Fontaine, M., J. Togelius, S. Nikolaidis et A. K. Hoover (2020). « Covariance Matrix Adaptation for the Rapid Illumination of Behavior Space ». In : *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico : Association for Computing Machinery, p. 94-102. ISBN : 9781450371285. DOI : 10.1145/3377930.3390232.

Forestier, S., R. Portelas, Y. Mollard et P.-Y. Oudeyer (2022). « Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning ». In : *Journal of Machine Learning Research* 23.152, p. 1-41.

Fu, J., A. Kumar, O. Nachum, G. Tucker et S. Levine (2020). « D4rl : Datasets for deep data-driven reinforcement learning ». In : *arXiv preprint :2004.07219*.

Fu, Z., T. Z. Zhao et C. Finn (2024). « Mobile aloha : Learning bimanual mobile manipulation with low-cost whole-body teleoperation ». In : *arXiv preprint :2401.02117*.

Fujimoto, S., H. Hoof et D. Meger (2018). « Addressing function approximation error in actor-critic methods ». In : *International conference on machine learning*. PMLR, p. 1587-1596.

Fuks, L., N. H. Awad, F. Hutter et M. Lindauer (2019). « An Evolution Strategy with Progressive Episode Lengths for Playing Games. » In : *IJCAI*, p. 1234-1240.

Gaier, A., A. Asteroth et J.-B. Mouret (2020). « Discovering representations for black-box optimization ». In : GECCO '20. Cancún, Mexico : Association for Computing Machinery, p. 103-111. ISBN : 9781450371285. DOI : 10.1145/3377930.3390221.

Gal, T. et J. Nedoma (1972). « Multiparametric linear programming ». In : *Management Science* 18.7, p. 406-422.

Galanos, T., A. Liapis, G. N. Yannakakis et R. Koenig (2021). « ARCH-Elites : Quality-diversity for urban design ». In : *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, p. 313-314.

Gašperov, B., M. Đurasević et D. Jakobovic (2024). « Finding Near-Optimal Portfolios With Quality-Diversity ». In : *arXiv preprint :2402.16118*.

Geijtenbeek, T., M. van de Panne et A. F. van der Stappen (nov. 2013). « Flexible muscle-based locomotion for bipedal creatures ». In : 32.6. ISSN : 0730-0301. DOI : 10.1145/2508363.2508399.

Gerhart, J. et M. Kirschner (2007). « The theory of facilitated variation ». In : *Proceedings of the National Academy of Sciences* 104.suppl_1, p. 8582-8589.

Gheribi, A., A. Pelton, E. Bélisle, S. Le Digabel et J.-P. Harvey (2018). « On the prediction of low-cost high entropy alloys using new thermodynamic multi-objective criteria ». In : *Acta Materialia* 161, p. 73-82.

Gomes, J., P. Urbano et A. L. Christensen (2013). « Evolution of swarm robotics systems with novelty search ». In : *Swarm Intelligence* 7, p. 115-144.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville et Y. Bengio (2014). « Generative adversarial nets ». In : *Advances in neural information processing systems* 27.

Gopnik, A., A. N. Meltzoff et P. K. Kuhl (1999). *The scientist in the crib : Minds, brains, and how children learn*. William Morrow & Co.

Goswami, A. et P. Vadakkepat (2018). *Humanoid Robotics : A Reference*. 1st. Springer Publishing Company, Incorporated. ISBN : 9400760450.

Grzes, M. (2017). « Reward shaping in episodic reinforcement learning ». In.

Gu, J., Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai et al. (2018). « Recent advances in convolutional neural networks ». In : *Pattern recognition* 77, p. 354-377.

Guan, Y., Y. Ren, S. E. Li, Q. Sun, L. Luo et K. Li (2020). « Centralized Cooperation for Connected and Automated Vehicles at Intersections by Proximal Policy Optimization ». In : *IEEE Transactions on Vehicular Technology* 69.11, p. 12597-12608. DOI : 10.1109/TVT.2020.3026111.

Gupta, A. et Y. Ong (déc. 2018). *Memetic Computation : The Mainspring of Knowledge Transfer in a Data-Driven Optimization Era*. ISBN : 978-3-030-02729-2.

Gupta, A., Y.-S. Ong et L. Feng (juin 2016). « Multifactorial Evolution : Toward Evolutionary Multitasking ». In : *IEEE Transactions on Evolutionary Computation* 20.3. Conference Name : IEEE Transactions on Evolutionary Computation, p. 343-357. ISSN : 1941-0026. DOI : 10.1109/TEVC.2015.2458037.

Gupta, A., L. Zhou, Y. Ong, Z. Chen et Y. Hou (mai 2022). « Half a Dozen Real-World Applications of Evolutionary Multitasking, and More ». In : *IEEE Computational Intelligence Magazine* 17, p. 49-66. DOI : 10.1109/MCI.2022.3155332.

Ha, D. et J. Schmidhuber (2018). « Recurrent world models facilitate policy evolution ». In : *Advances in neural information processing systems* 31.

Ha, S., P. Xu, Z. Tan, S. Levine et J. Tan (2020). « Learning to walk in the real world with minimal human effort ». In : *arXiv preprint :2002.08550*.

Haarnoja, T., A. Zhou, P. Abbeel et S. Levine (oct. 2018). « Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor ». In : *Proceedings of the 35th International Conference on Machine Learning*. Sous la dir. de J. Dy et A. Krause. T. 80. Proceedings of Machine Learning Research. PMLR, p. 1861-1870.

Hansen, N., S. Muller et P. Koumoutsakos (fév. 2003). « Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES) ». In : *Evolutionary computation* 11, p. 1-18. DOI : 10.1162/106365603321828970.

Hao, X., R. Qu et J. Liu (2021). « A Unified Framework of Graph-Based Evolutionary Multitasking Hyper-Heuristic ». In : *IEEE Transactions on Evolutionary Computation* 25.1, p. 35-47. DOI : 10.1109/TEVC.2020.2991717.

Hart, P. E., N. J. Nilsson et B. Raphael (1968). « A Formal Basis for the Heuristic Determination of Minimum Cost Paths ». In : *IEEE Transactions on Systems Science and Cybernetics* 4.2, p. 100-107. DOI : 10.1109/TSSC.1968.300136.

Hauert, S., J.-C. Zufferey et D. Floreano (2009). « Evolved swarming without positioning information : an application in aerial communication relay ». In : *Autonomous Robots* 26, p. 21-32.

Hegde, S., S. Batra, K. Zentner et G. Sukhatme (2024). « Generating behaviorally diverse policies with latent diffusion models ». In : *Advances in Neural Information Processing Systems* 36.

Henderson, P. (2018). *Reproducibility and reusability in deep reinforcement learning*. McGill University (Canada).

Herty, M. et A. Klar (2003). « Modeling, Simulation, and Optimization of Traffic Flow Networks ». In : *SIAM Journal on Scientific Computing* 25.3, p. 1066-1087. DOI : 10.1137/S106482750241459X.

Heuillet, A., F. Couthouis et N. Diaz-Rodriguez (2021). « Explainability in deep reinforcement learning ». In : *Knowledge-Based Systems* 214, p. 106685.

Hill, A., J. Laneurit, R. Lenain et E. Lucet (2020). « Online gain setting method for path tracking using CMA-ES : Application to off-road mobile robot control ». In : *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 7697-7702. DOI : 10.1109/IROS45743.2020.9340830.

Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath et B. Kingsbury (2012). « Deep Neural Networks for Acoustic Modeling in Speech Recognition : The Shared Views of Four Research Groups ». In : *IEEE Signal Processing Magazine* 29.6, p. 82-97. DOI : 10.1109/MSP.2012.2205597.

Hinton, G. E., S. J. Nowlan et al. (1987). « How learning can guide evolution ». In : *Complex systems* 1.3, p. 495-502.

Ho, J. et S. Ermon (2016). « Generative Adversarial Imitation Learning ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de D. Lee, M. Sugiyama, U. Luxburg, I. Guyon et R. Garnett. T. 29. Curran Associates, Inc.

Holland, J. H. (1992). « Genetic algorithms ». In : *Scientific american* 267.1, p. 66-73.

Hollerbach, J., W. Khalil et M. Gautier (2008). « Model Identification ». In : *Springer Handbook of Robotics*. Sous la dir. de B. Siciliano et O. Khatib. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 321-344. DOI : 10.1007/978-3-540-30301-5_15.

Hoppitt, W. J., G. R. Brown, R. Kendal, L. Rendell, A. Thornton, M. M. Webster et K. N. Laland (2008). « Lessons from animal teaching ». In : *Trends in ecology & evolution* 23.9, p. 486-493.

Hornik, K., M. Stinchcombe et H. White (1989). « Multilayer feedforward networks are universal approximators ». In : *Neural networks* 2.5, p. 359-366.

Hospedales, T., A. Antoniou, P. Micaelli et A. Storkey (2021). « Meta-learning in neural networks : A survey ». In : *IEEE transactions on pattern analysis and machine intelligence* 44.9, p. 5149-5169.

Houthooft, R., Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho et P. Abbeel (2018). « Evolved policy gradients ». In : *Advances in Neural Information Processing Systems* 31.

Huang, S., J. Zhong et W.-J. Yu (2021). « Surrogate-Assisted Evolutionary Framework with Adaptive Knowledge Transfer for Multi-Task Optimization ». In : *IEEE Transactions on Emerging Topics in Computing* 9.4, p. 1930-1944. DOI : 10.1109/TETC.2019.2945775.

Huynh Thi Thanh, B., L. Van Cuong, T. B. Thang et N. H. Long (2023a). « Ensemble Multifactorial Evolution With Biased Skill-Factor Inheritance for Many-Task Optimization ». In : *IEEE Transactions on Evolutionary Computation* 27.6, p. 1735-1749. DOI : 10.1109/TEVC.2022.3227120.

— (déc. 2023b). « Ensemble Multifactorial Evolution With Biased Skill-Factor Inheritance for Many-Task Optimization ». In : *IEEE Transactions on Evolutionary Computation* 27.6. Conference Name : IEEE Transactions on Evolutionary Computation, p. 1735-1749. ISSN : 1941-0026. DOI : 10.1109/TEVC.2022.3227120.

Hwangbo, J., J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun et M. Hutter (2019). « Learning agile and dynamic motor skills for legged robots ». In : *Science Robotics* 4.26, eaau5872.

Iida, F. et A. J. Ijspeert (2016). « Biologically inspired robotics ». In : *Springer Handbook of Robotics*, p. 2015-2034.

Ijspeert, A. J., J. Nakanishi, H. Hoffmann, P. Pastor et S. Schaal (2013). « Dynamical movement primitives : learning attractor models for motor behaviors ». In : *Neural computation* 25.2, p. 328-373.

Iscen, A., K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani et V. Vanhoucke (2018). « Policies modulating trajectory generators ». In : *Conference on Robot Learning*. PMLR, p. 916-926.

Jacobson, D. et D. Mayne (1970). *Differential Dynamic Programming*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company. ISBN : 9780444000705.

Jegorova, M., S. Doncieux et T. M. Hospedales (2020). « Behavioral repertoire via generative adversarial policy networks ». In : *IEEE Transactions on Cognitive and Developmental Systems*.

Jumper, J., R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Židek, A. Potapenko et al. (2021). « Highly accurate protein structure prediction with AlphaFold ». In : *Nature* 596.7873, p. 583-589.

Kaelbling, L. P., M. L. Littman et A. W. Moore (1996). « Reinforcement learning : A survey ». In : *Journal of artificial intelligence research* 4, p. 237-285.

Karras, T., S. Laine, M. Aittala, J. Hellsten, J. Lehtinen et T. Aila (2020). « Analyzing and improving the image quality of stylegan ». In : *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, p. 8110-8119.

Kaufmann, E., L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun et D. Scaramuzza (2023). « Champion-level drone racing using deep reinforcement learning ». In : *Nature* 620.7976, p. 982-987.

Kaushik, R., T. Anne et J.-B. Mouret (2020). « Fast Online Adaptation in Robotics through Meta-Learning Embeddings of Simulated Priors ». In : *IEEE/RSJ IROS*. IEEE, p. 5269-5276. DOI : 10.1109/IROS45743.2020.9341462.

Kelly, M., C. Sidrane, K. Driggs-Campbell et M. J. Kochenderfer (2019). « Hg-dagger : Interactive imitation learning with human experts ». In : *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, p. 8077-8083.

Kheddar, A. et al. (2019). « Humanoid robots in aircraft manufacturing : The Airbus use cases ». In : *IEEE RA Magazine* 26.4, p. 30-45.

Kingma, D. P. et J. Ba (2014). « Adam : A method for stochastic optimization ». In : *arXiv preprint :1412.6980*.

Kober, J. et J. Peters (2009). « Learning motor primitives for robotics ». In : *2009 IEEE International Conference on Robotics and Automation*, p. 2112-2118. DOI : 10.1109/ROBOT.2009.5152577.

Kobilarov, M. (2012). « Cross-entropy motion planning ». In : *The International Journal of Robotics Research* 31.7, p. 855-871. DOI : 10.1177/0278364912444543.

Koza, J. R. et al. (1994). *Genetic programming II*. T. 17. MIT press Cambridge.

Kristensen, J. T. et P. Burelli (2020). « Strategies for Using Proximal Policy Optimization in Mobile Puzzle Games ». In : *Proceedings of the 15th International Conference on the Foundations of Digital Games*. FDG '20. Bugibba, Malta : Association for Computing Machinery. ISBN : 9781450388078. DOI : 10.1145/3402942.3402944.

Krizhevsky, A., I. Sutskever et G. E. Hinton (2012). « Imagenet classification with deep convolutional neural networks ». In : *Advances in neural information processing systems* 25.

Kroemer, O., S. Niekum et G. Konidaris (2021). « A review of robot learning for manipulation : Challenges, representations, and algorithms ». In : *Journal of machine learning research* 22.30, p. 1-82.

Kumar, A. S. et Z. Ahmad (2012). « MODEL PREDICTIVE CONTROL (MPC) AND ITS CURRENT ISSUES IN CHEMICAL ENGINEERING ». In : *Chemical Engineering Communications* 199.4, p. 472-511. DOI : 10.1080/00986445.2011.592446.

Kurtz, V., H. Li, P. M. Wensing et H. Lin (2021). « Mini Cheetah, the Falling Cat : A Case Study in Machine Learning and Trajectory Optimization for Robot Acrobatics ». In : *CoRR* abs/2109.04424. arXiv : 2109.04424.

Ladosz, P., L. Weng, M. Kim et H. Oh (2022). « Exploration in deep reinforcement learning : A survey ». In : *Information Fusion* 85, p. 1-22.

LaValle, S. (2006). « Planning Algorithms ». In : *Cambridge University Press google schola* 2, p. 3671-3678.

Laversanne-Finot, A., A. Péré et P.-Y. Oudeyer (2021). « Intrinsically motivated exploration of learned goal spaces ». In : *Frontiers in neurorobotics* 14, p. 555271.

LeCun, Y., Y. Bengio et G. Hinton (2015). « Deep learning ». In : *nature* 521.7553, p. 436-444.

LeCun, Y., S. Chopra, R. Hadsell, M. Ranzato et F. Huang (2006). « A tutorial on energy-based learning ». In : *Predicting structured data* 1.0.

Lee, J., J. Hwangbo, L. Wellhausen, V. Koltun et M. Hutter (2020). « Learning quadrupedal locomotion over challenging terrain ». In : *Science robotics* 5.47, eabc5986.

Lehman, J., K. O. Stanley et al. (2008). « Exploiting open-endedness to solve problems through the search for novelty. » In : *ALIFE*, p. 329-336.

Lehman, J. et K. O. Stanley (juin 2011a). « Abandoning Objectives : Evolution Through the Search for Novelty Alone ». In : *Evolutionary computation* 19, p. 189-223. DOI : 10.1162/EVCO_a_00025.

— (2011b). « Evolving a Diversity of Virtual Creatures through Novelty Search and Local Competition ». In : *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. Dublin, Ireland : Association for Computing Machinery, p. 211-218. ISBN : 9781450305570. DOI : 10.1145/2001576.2001606.

Levine, S., C. Finn, T. Darrell et P. Abbeel (2016). « End-to-end training of deep visuomotor policies ». In : *Journal of Machine Learning Research* 17.39, p. 1-40.

Levine, S. et V. Koltun (2013). « Guided policy search ». In : *International conference on machine learning*. PMLR, p. 1-9.

— (2014). « Learning complex neural network policies with trajectory optimization ». In : *International Conference on Machine Learning*. PMLR, p. 829-837.

Li, Q. et al. (2017). « A minimized falling damage method for humanoid robots ». In : *IJARS* 14.5.

Li, Y., J. Song et S. Ermon (2017). « Infogail : Interpretable imitation learning from visual demonstrations ». In : *Advances in neural information processing systems* 30.

Liang, J., K. Qiao, M. Yuan, K. Yu, B. Qu, S. Ge, Y. Li et G. Chen (mars 2020). « Evolutionary multi-task optimization for parameters extraction of photovoltaic models ». In : *Energy Conversion and Management* 207, p. 112509. DOI : 10.1016/j.enconman.2020.112509.

Liang, Z., X. Xu, L. Liu, Y. Tu et Z. Zhu (2022). « Evolutionary Many-Task Optimization Based on Multisource Knowledge Transfer ». In : *IEEE Transactions on Evolutionary Computation* 26.2, p. 319-333. DOI : 10.1109/TEVC.2021.3101697.

Liapis, A., G. N. Yannakakis et J. Togelius (2015). « Constrained novelty search : A study on game content generation ». In : *Evolutionary computation* 23.1, p. 101-129.

Liaw, R.-T. et C.-K. Ting (juin 2017). « Evolutionary many-tasking based on biocoenosis through symbiosis : A framework and benchmark problems ». In : *2017 IEEE Congress on Evolutionary Computation (CEC)*, p. 2266-2273. DOI : 10.1109/CEC.2017.7969579.

— (juill. 2019). « Evolutionary Manytasking Optimization Based on Symbiosis in Biocoenosis ». en. In : *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01. Number : 01, p. 4295-4303. ISSN : 2374-3468. DOI : 10.1609/aaai.v33i01.33014295.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver et D. Wierstra (2015a). « Continuous control with deep reinforcement learning ». In : *arXiv preprint :1509.02971*.

— (2015b). « Continuous control with deep reinforcement learning ». In : *arXiv preprint :1509.02971*.

Lin, S. et P. A. Beling (2021). « An end-to-end optimal trade execution framework based on proximal policy optimization ». In : *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, p. 4548-4554.

Lipson, H. et J. B. Pollack (2000). « Automatic design and manufacture of robotic lifeforms ». In : *Nature* 406.6799, p. 974-978.

Liu, D., Y. Lin et V. Kapila (2021). « A Rollover Strategy for Wrist Damage Reduction in a Forward Falling Humanoid ». In : *IEEE International Conference on Mechatronics and Automation (ICMA)*. DOI : 10.1109/ICMA52036.2021.9512722.

Liu, G., L. Zhao, F. Yang, J. Bian, T. Qin, N. Yu et T.-Y. Liu (2019). « Trust region evolution strategies ». In : *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 33. 01, p. 4352-4359.

Liu, J., P. Li, G. Wang, Y. Zha, J. Peng et G. Xu (2020). « A Multitasking Electric Power Dispatch Approach With Multi-Objective Multifactorial Optimization Algorithm ». In : *IEEE Access* 8, p. 155902-155911. DOI : 10.1109/ACCESS.2020.3018484.

Lu, J. Z. (2015). « Closing the gap between planning and control : A multiscale MPC cascade approach ». In : *Annual Reviews in Control* 40, p. 3-13. ISSN : 1367-5788. DOI : https://doi.org/10.1016/j.arcontrol.2015.09.016.

Lu, T., D. Pal et M. Pal (13–15 May 2010). « Contextual Multi-Armed Bandits ». In : *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Sous la dir. d'Y. W. Teh et M. Titterington. T. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy : PMLR, p. 485-492.

Lupuleac, S., T. Pogarskaia, M. Churilova, M. Kokkolaras et E. Bonhomme (2020). « Optimization of fastener pattern in airframe assembly ». In : *Assembly Automation* 40.5, p. 723-733.

MacAlpine, P., M. Depinet et P. Stone (2015). « UT Austin Villa 2014 : RoboCup 3D simulation league champion via overlapping layered learning ». In : *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 29. 1.

Macé, V., R. Boige, F. Chalumeau, T. Pierrot, G. Richard et N. Perrin-Gilbert (2023). « The Quality-Diversity Transformer : Generating Behavior-Conditioned Trajectories with Decision Transformers ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Lisbon, Portugal : Association for Computing Machinery, p. 1221-1229. ISBN : 9798400701191. DOI : 10.1145/3583131.3590433.

Majid, A. Y., S. Saaybi, V. Francois-Lavet, R. V. Prasad et C. Verhoeven (2023). « Deep reinforcement learning versus evolution strategies : a comparative survey ». In : *IEEE Transactions on Neural Networks and Learning Systems*.

Maneewongvatana, S. et D. M. Mount (1999). « Analysis of approximate nearest neighbor searching with clustered point sets ». In : *arXiv preprint cs/9901013*.

Mania, H., A. Guy et B. Recht (2018). « Simple random search of static linear policies is competitive for reinforcement learning ». In : *Advances in neural information processing systems* 31.

Marsden, A. L., J. A. Feinstein et C. A. Taylor (2008). « A computational framework for derivative-free optimization of cardiovascular geometries ». In : *Computer methods in applied mechanics and engineering* 197.21-24, p. 1890-1905.

Mastalli, C. et al. (2020). « Crocoddyl : An Efficient and Versatile Framework for Multi-Contact Optimal Control ». In : *IEEE ICRA*.

Matthews, B. (1975). « Comparison of the predicted and observed secondary structure of T4 phage lysozyme ». In : *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405.2, p. 442-451. ISSN : 0005-2795. DOI : https://doi.org/10.1016/0005-2795(75)90109-9.

Medina, A., M. Richey, M. Mueller et J. Schrum (2023). « Evolving Flying Machines in Minecraft Using Quality Diversity ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 1418-1426.

Miki, T., J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun et M. Hutter (2022). « Learning robust perceptive locomotion for quadrupedal robots in the wild ». In : *Science Robotics* 7.62, eabk2822.

Min, A. T. W., Y.-S. Ong, A. Gupta et C.-K. Goh (2019). « Multiproblem Surrogates : Transfer Evolutionary Multiobjective Optimization of Computationally Expensive Problems ». In : *IEEE Transactions on Evolutionary Computation* 23.1, p. 15-28. DOI : 10.1109/TEVC.2017.2783441.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver et K. Kavukcuoglu (2016a). « Asynchronous methods for deep reinforcement learning ». In : *International conference on machine learning*. PMLR, p. 1928-1937.

— (2016b). « Asynchronous methods for deep reinforcement learning ». In : *International conference on machine learning*. PMLR, p. 1928-1937.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra et M. Riedmiller (2013). « Playing atari with deep reinforcement learning ». In : *arXiv preprint :1312.5602*.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al. (2015). « Human-level control through deep reinforcement learning ». In : *nature* 518.7540, p. 529-533.

Moerland, T. M., J. Broekens, A. Plaat, C. M. Jonker et al. (2023). « Model-based reinforcement learning : A survey ». In : *Foundations and Trends® in Machine Learning* 16.1, p. 1-118.

Mouret, J.-B. (2011). « Novelty-based multiobjectivization ». In : *New Horizons in Evolutionary Robotics : Extended Contributions from the 2009 EvoDeRob Workshop*. Springer, p. 139-154.

— (2020). « Evolving the behavior of machines : from micro to macroevolution ». In : *Iscience* 23.11.

— (2024). *Large language models help computer programs to evolve.*

Mouret, J.-B. et J. Clune (2015). *Illuminating search spaces by mapping elites*. DOI : 10.48550/ARXIV.1504.04909.

Mouret, J.-B. et S. Doncieux (2009). « Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity ». In : *2009 IEEE congress on evolutionary computation*. IEEE, p. 1161-1168.

— (2012). « Encouraging behavioral diversity in evolutionary robotics : An empirical study ». In : *Evolutionary computation* 20.1, p. 91-133.

Mouret, J.-B. et G. Maguire (2020). « Quality Diversity for Multi-Task Optimization ». In : *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico : Association for Computing Machinery, p. 121-129. ISBN : 9781450371285. DOI : 10.1145/3377930.3390203.

Nagabandi, A., I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine et C. Finn (2018). « Learning to adapt in dynamic, real-world environments through meta-reinforcement learning ». In : *arXiv preprint 1803.11347*.

— (2019). « Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning ». In : *ICLR*.

Nagabandi, A., G. Kahn, R. S. Fearing et S. Levine (2018). « Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning ». In : *2018 IEEE International Conference on Robotics and Automation (ICRA)*, p. 7559-7566. DOI : 10.1109/ICRA.2018.8463189.

Nagarajan, P., G. Warnell et P. Stone (2018). « Deterministic implementations for reproducibility in deep reinforcement learning ». In : *arXiv preprint :1809.05676*.

Naredo, E. et L. Trujillo (2013). « Searching for novel clustering programs ». In : *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, p. 1093-1100.

Nilsson, O. et A. Cully (juin 2021). « Policy gradient assisted MAP-Elites ». en. In : *Proceedings of the Genetic and Evolutionary Computation Conference*. Lille France : ACM, p. 866-875. ISBN : 978-1-4503-8350-9. DOI : 10.1145/3449639.3459304.

Nolfi, S. et D. Floreano (2000). *Evolutionary robotics : The biology, intelligence, and technology of self-organizing machines*. MIT press.

Norman, B. et J. Clune (2023). *First-Explore, then Exploit : Meta-Learning Intelligent Exploration*. arXiv : 2307.02276 [cs.LG].

Oudeyer, P.-Y., F. Kaplan et V. V. Hafner (2007). « Intrinsic motivation systems for autonomous mental development ». In : *IEEE transactions on evolutionary computation* 11.2, p. 265-286.

Paolo, G., A. Coninx, S. Doncieux et A. Laflaquière (2021). « Sparse reward exploration via novelty search and emitters ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 154-162.

Pappas, I., D. Kenefake, B. Burnak, S. Avraamidou, H. S. Ganesh, J. Katz, N. A. Diangelakis et E. N. Pistikopoulos (2021). « Multiparametric Programming in Process Systems Engineering : Recent Developments and Path Forward ». In : *Frontiers in Chemical Engineering* 2. ISSN : 2673-2718. DOI : 10.3389/fceng.2020.620168.

Parisi, G. I., R. Kemker, J. L. Part, C. Kanan et S. Wermter (2019). « Continual lifelong learning with neural networks : A review ». In : *Neural networks* 113, p. 54-71.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai et S. Chintala (2019). « PyTorch : An Imperative Style, High-Performance Deep Learning Library ». In : *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., p. 8024-8035.

Pearce, M. et J. Branke (2018). « Continuous multi-task Bayesian Optimisation with correlation ». In : *European Journal of Operational Research* 270.3, p. 1074-1085. ISSN : 0377-2217. DOI : https://doi.org/10.1016/j.ejor.2018.03.017.

Pearl, J. (1984). *Heuristics : intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc.

Perez-Liebana, D., C. Guerrero-Romero, A. Dockhorn, L. Xu, J. Hurtado et D. Jeurissen (2021). « Generating diverse and competitive play-styles for strategy games ». In : *2021 IEEE Conference on Games (CoG)*. IEEE, p. 1-8.

Peters, J. et S. Schaal (2008). « Natural actor-critic ». In : *Neurocomputing* 71.7-9, p. 1180-1190.

Pfeifer, R. et J. Bongard (2006). *How the body shapes the way we think : a new view of intelligence*. MIT press.

Pfeifer, R., M. Lungarella et F. Iida (2007). « Self-organization, embodiment, and biologically inspired robotics ». In : *science* 318.5853, p. 1088-1093.

Pinciroli, L., P. Baraldi, G. Ballabio, M. Compare et E. Zio (2021). « Deep Reinforcement Learning Based on Proximal Policy Optimization for the Maintenance of a Wind Farm with Multiple Crews ». In : *Energies* 14.20. ISSN : 1996-1073. DOI : 10.3390/en14206743.

Piryonesi, s. M. et M. Tavakolan (jan. 2017). « A mathematical programming model for solving cost-safety optimization (CSO) problems in the maintenance of structures ». In : *KSCE Journal of Civil Engineering* 21. DOI : 10.1007/s12205-017-0531-z.

Pistikopoulos, E. N., V. Dua, N. A. Bozinis, A. Bemporad et M. Morari (juill. 2000). « On-line optimization via off-line parametric optimization tools ». In : *Computers & Chemical Engineering* 24.2, p. 183-188. ISSN : 0098-1354. DOI : 10.1016/S0098-1354(00)00510-X.

— (2002). « On-line optimization via off-line parametric optimization tools ». In : *Computers and Chemical Engineering* 26.2, p. 175-185. ISSN : 0098-1354. DOI : https://doi.org/10.1016/S0098-1354(01)00739-6.

Polverini, M. P. et al. (2021). « Agile Actions with a Centaur-Type Humanoid : A Decoupled Approach ». In : *IEEE ICRA*. DOI : 10.1109/ICRA48506.2021.9561239.

Pomerleau, D. A. (1988). « Alvinn : An autonomous land vehicle in a neural network ». In : *Advances in neural information processing systems* 1.

Poppinga, J., N. Vaskevicius, A. Birk et K. Pathak (2008). « Fast plane detection and polygonalization in noisy 3D range images ». In : *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, p. 3378-3383.

Pourchot, A. et O. Sigaud (2019). « CEM-RL : Combining evolutionary and gradient-based methods for policy search ». In : *7th International Conference on Learning Representations, ICLR 2019*.

Pugh, J. K., L. B. Soros et K. O. Stanley (2016). « Quality diversity : A new frontier for evolutionary computation ». In : *Frontiers in Robotics and AI* 3, p. 202845.

Pugh, J. K., L. B. Soros, P. A. Szerlip et K. O. Stanley (2015). « Confronting the challenge of quality diversity ». In : *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, p. 967-974.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever et al. (2019). « Language models are unsupervised multitask learners ». In : *OpenAI blog* 1.8, p. 9.

Raffin, A., A. Hill, A. Gleave, A. Kanervisto, M. Ernestus et N. Dormann (2021). « Stable-Baselines3 : Reliable Reinforcement Learning Implementations ». In : *Journal of Machine Learning Research* 22.268, p. 1-8.

Rakicevic, N., A. Cully et P. Kormushev (2021). « Policy manifold search : Exploring the manifold hypothesis for diversity-based neuroevolution ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 901-909.

Ramuzat, N. et al. (2020). « Actuator Model, Identification and Differential Dynamic Programming for a TALOS Humanoid Robot ». In : *European Control Conference (ECC)*, p. 724-730. DOI : 10.23919/ECC51009.2020.9143817.

Ravichandar, H., A. S. Polydoros, S. Chernova et A. Billard (2020). « Recent advances in robot learning from demonstration ». In : *Annual review of control, robotics, and autonomous systems* 3, p. 297-330.

Rechenberg, I. (1972). *Evolutionstrategie : Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Stuttgart : Fromman-Hozlboog Verlag.

Risi, S., C. E. Hughes et K. O. Stanley (2010). « Evolving plastic neural networks with novelty search ». In : *Adaptive Behavior* 18.6, p. 470-491.

Rolf, M., J. J. Steil et M. Gienger (2010). « Goal Babbling Permits Direct Learning of Inverse Kinematics ». In : *IEEE Transactions on Autonomous Mental Development* 2.3, p. 216-229. DOI : 10.1109/TAMD.2010.2062511.

Romero, A., S. Sun, P. Foehn et D. Scaramuzza (2022). « Model predictive contouring control for time-optimal quadrotor flight ». In : *IEEE Transactions on Robotics* 38.6, p. 3340-3356.

Ross, S., G. Gordon et D. Bagnell (2011). « A reduction of imitation learning and structured prediction to no-regret online learning ». In : *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop et Conference Proceedings, p. 627-635.

Rouxel, Q., S. Ivaldi et J.-B. Mouret (2023). « Multi-Contact Whole Body Force Control for Position-Controlled Robots ». In : *arXiv preprint :2312.16465*.

Rubinstein, R. Y. et D. P. Kroese (2004a). *The cross-entropy method : a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. T. 133. Springer.

— (2004b). *The Cross-Entropy Method : A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*.

Rückstiess, T., F. Sehnke, T. Schaul, D. Wierstra, Y. Sun et J. Schmidhuber (2010). « Exploring parameter space in reinforcement learning ». In : *Paladyn* 1, p. 14-24.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg et L. Fei-Fei (2015). « ImageNet Large Scale Visual Recognition Challenge ». In : *International Journal of Computer Vision (IJCV)* 115.3, p. 211-252. DOI : 10.1007/s11263-015-0816-y.

Sagarna, R. et Y.-S. Ong (2016). « Concurrently searching branches in software tests generation through multitask evolution ». In : *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, p. 1-8. DOI : 10.1109/SSCI.2016.7850040.

Salimans, T., J. Ho, X. Chen, S. Sidor et I. Sutskever (2017). « Evolution strategies as a scalable alternative to reinforcement learning ». In : *arXiv preprint :1703.03864*.

Salman, R. et V. Kecman (2012). « Regression as classification ». In : *2012 Proceedings of IEEE Southeastcon*. IEEE, p. 1-6.

Samy, V., K. Bouyarmane et A. Kheddar (2017). « QP-based adaptive-gains compliance control in humanoid falls ». In : *IEEE ICRA*. DOI : 10.1109/ICRA.2017.7989553.

Samy, V., S. Caron, K. Bouyarmane et A. Kheddar (2017). « Post-impact adaptive compliance for humanoid falls using predictive control of a reduced model ». In : *IEEE-RAS Humanoids*. DOI : 10.1109/HUMANOIDS.2017.8246942.

Samy, V. et A. Kheddar (2015). « Falls control using posture reshaping and active compliance ». In : *IEEE-RAS Humanoids*. DOI : 10.1109/HUMANOIDS.2015.7363469.

Schaal, S. (1996a). « Learning from Demonstration ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de M. Mozer, M. Jordan et T. Petsche. T. 9. MIT Press.

— (1996b). « Learning from demonstration ». In : *Advances in neural information processing systems* 9.

Schulman, J., S. Levine, P. Abbeel, M. Jordan et P. Moritz (2015). « Trust region policy optimization ». In : *International conference on machine learning*. PMLR, p. 1889-1897.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford et O. Klimov (2017). « Proximal Policy Optimization Algorithms ». In : *CoRR* abs/1707.06347.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. 1995 – 2^{nd} edition. New-York : John Wiley & Sons.

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams et N. de Freitas (2016). « Taking the Human Out of the Loop : A Review of Bayesian Optimization ». In : *Proceedings of the IEEE* 104.1, p. 148-175. DOI : 10.1109/JPROC.2015.2494218.

Siciliano, B., O. Khatib et T. Kröger (2008). *Springer handbook of robotics*. T. 200. Springer.

Siegwart, R., I. R. Nourbakhsh et D. Scaramuzza (2011). *Introduction to autonomous mobile robots*. MIT press.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al. (2016). « Mastering the game of Go with deep neural networks and tree search ». In : *nature* 529.7587, p. 484-489.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al. (2017). « Mastering the game of go without human knowledge ». In : *nature* 550.7676, p. 354-359.

Skinner, B. (1938). « The behavior of organisms : an experimental analysis. » In.

Solnik, B., D. Golovin, G. Kochanski, J. E. Karro, S. Moitra et D. Sculley (2017). « Bayesian Optimization for a Better Dessert ». In : *Proceedings of the 2017 NIPS Workshop on Bayesian Optimization*. The workshop is BayesOpt 2017 NIPS Workshop on Bayesian Optimization December 9, 2017, Long Beach, USA. December 9, 2017, Long Beach, USA.

Soltoggio, A., P. Durr, C. Mattiussi et D. Floreano (2007). « Evolving neuromodulatory topologies for reinforcement learning-like problems ». In : *2007 IEEE Congress on evolutionary computation*. IEEE, p. 2471-2478.

Song, Y., A. Romero, M. Müller, V. Koltun et D. Scaramuzza (2023). « Reaching the limit in autonomous racing : Optimal control versus reinforcement learning ». In : *Science Robotics* 8.82, eadg1462.

Spitz, J. et al. (2017). « Trial-and-error learning of repulsors for humanoid QP-based whole-body control ». In : *IEEE Humanoids*.

Stadie, B. C., G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel et I. Sutskever (2018). « Some considerations on learning to explore via meta-reinforcement learning ». In : *arXiv preprint 1803.01118*.

Stanley, K. O. et J. Lehman (2015). *Why greatness cannot be planned : The myth of the objective*. Springer.

Stanley, K. O. et R. Miikkulainen (2002). « Evolving Neural Networks through Augmenting Topologies ». In : *Evolutionary Computation* 10.2, p. 99-127.

Stasse, O. et al. (2017). « TALOS : A new humanoid research platform targeted for industrial applications ». In : *IEEE Humanoids*. DOI : 10.1109/HUMANOIDS.2017.8246947.

Stillings, N. A., C. H. Chase, S. E. Weisler, M. H. Feinstein et E. L. Rissland (1995). *Cognitive science : An introduction*. MIT press.

Stulp, F. et O. Sigaud (2013). « Robot skill learning : From reinforcement learning to evolution strategies ». In : *Paladyn, Journal of Behavioral Robotics* 4.1, p. 49-61.

Sugihara, T., Y. Nakamura et H. Inoue (2002). « Real-time humanoid motion generation through ZMP manipulation based on inverted pendulum control ». In : *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. T. 2, 1404-1409 vol.2. DOI : 10.1109/ROBOT.2002.1014740.

Sun, H., L. Yang, Y. Gu, J. Pan, F. Wan et C. Song (2023). « Bridging Locomotion and Manipulation Using Reconfigurable Robotic Limbs via Reinforcement Learning ». In : *Biomimetics* 8.4. ISSN : 2313-7673. DOI : 10.3390/biomimetics8040364.

Surana, S., B. Lim et A. Cully (2023). « Efficient Learning of Locomotion Skills through the Discovery of Diverse Environmental Trajectory Generator Priors ». In : *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, p. 12134-12141.

Sutton, R. S. et A. G. Barto (1998). *Reinformcent Learning : An Introduction*.

Sutton, R. S., D. McAllester, S. Singh et Y. Mansour (1999). « Policy gradient methods for reinforcement learning with function approximation ». In : *Advances in neural information processing systems* 12.

Tan, J., K. Liu et G. Turk (2011). « Stable Proportional-Derivative Controllers ». In : *IEEE Computer Graphics and Applications* 31.4. ISSN : 0272-1716. DOI : 10.1109/MCG.2011.30.

Tan, J., T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez et V. Vanhoucke (2018). « Sim-to-real : Learning agile locomotion for quadruped robots ». In : *arXiv preprint :1804.10332*.

Tanaka, M. et K. Sekiyama (2023). « Human-Robot Imitation Learning of Movement for Embodiment Gap ». In : *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, p. 1733-1738.

Tempez, V. (2022). « Apprentissage d'une loi de commande optimale d'un petit quadrotor pour le vol dans des tuyaux cylindriques ». 2022LORR0072. Thèse de doct.

Theodorou, E., J. Buchli et S. Schaal (13–15 May 2010). « Learning Policy Improvements with Path Integrals ». In : *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Sous la dir. d'Y. W. Teh et M. Titterington. T. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy : PMLR, p. 828-835.

Thorndike, E. (1898). « Some experiments on animal intelligence ». In : *Science* 7.181, p. 818-824.

Thornton, A. et N. J. Raihani (2008). « The evolution of teaching ». In : *Animal behaviour* 75.6, p. 1823-1836.

Thrun, S. (2002). « Probabilistic robotics ». In : *Communications of the ACM* 45.3, p. 52-57.

Thrun, S. et T. M. Mitchell (1995). « Lifelong robot learning ». In : *Robotics and autonomous systems* 15.1-2, p. 25-46.

Tjanaka, B., M. Fontaine, D. H. Lee, A. Kalkar et S. Nikolaidis (2023). « Training diverse high-dimensional controllers by scaling covariance matrix adaptation map-annealing ». In : *IEEE Robotics and Automation Letters*.

Tjanaka, B., M. Fontaine, J. Togelius et S. Nikolaidis (2022). « Approximating gradients for differentiable quality diversity in reinforcement learning ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '22. Boston, Massachusetts : Association for Computing Machinery, p. 1102-1111. ISBN : 9781450392372. DOI : 10.1145/3512290.3528705.

Todorov, E., T. Erez et Y. Tassa (2012). « Mujoco : A physics engine for model-based control ». In : *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, p. 5026-5033.

Toneva, M., A. Sordoni, R. T. d. Combes, A. Trischler, Y. Bengio et G. J. Gordon (2018). « An empirical study of example forgetting during deep neural network learning ». In : *arXiv preprint :1812.05159*.

Torabi, F., G. Warnell et P. Stone (2018). « Behavioral cloning from observation ». In : *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, p. 4950-4957.

Torgo, L. et J. Gama (1996). « Regression by classification ». In : *Advances in Artificial Intelligence : 13th Brazilian Symposium on Artificial Intelligence, SBIA'96 Curitiba, Brazil, October 23–25, 1996 Proceedings 13*. Springer, p. 51-60.

Urquhart, N., E. Hart et W. Hutcheson (2020). « Using MAP-Elites to support policy making around Workforce Scheduling and Routing ». In : *at-Automatisierungstechnik* 68.2, p. 110-117.

Urzelai, J. et D. Floreano (2001). « Evolution of adaptive synapses : Robots with fast adaptive behavior in new environments ». In : *Evolutionary computation* 9.4, p. 495-524.

Vapnik, V. et A. Vashist (2009). « A new learning paradigm : Learning using privileged information ». In : *Neural networks* 22.5-6, p. 544-557.

Vassiliades, V., K. Chatzilygeroudis et J.-B. Mouret (août 2017). « Using Centroidal Voronoi Tessellations to Scale Up the Multi-dimensional Archive of Phenotypic Elites Algorithm ». In : *IEEE Transactions on Evolutionary Computation* PP, p. 1-1. DOI : 10.1109/TEVC.2017.2735550.

Vassiliades, V. et J.-B. Mouret (2018). « Discovering the Elite Hypervolume by Leveraging Interspecies Correlation ». In : *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan : Association for Computing Machinery, p. 149-156. ISBN : 9781450356183. DOI : 10.1145/3205455.3205602.

Vereshchagin, A. F. (1989). « Modeling and control of motion of manipulational robots ». In : *Soviet journal of computer and systems sciences* 27.5, p. 29-38.

Viana, B. M., L. T. Pereira et C. F. Toledo (2022). « Illuminating the Space of Dungeon Maps, Locked-door Missions and Enemy Placement Through MAP-Elites ». In : *arXiv preprint :2202.09301*.

Vinyals, O., I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev et al. (2019). « Grandmaster level in StarCraft II using multi-agent reinforcement learning ». In : *Nature* 575.7782, p. 350-354.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey,

İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt et SciPy 1.0 Contributors (2020). « SciPy 1.0 : Fundamental Algorithms for Scientific Computing in Python ». In : *Nature Methods* 17, p. 261-272. DOI : 10.1038/s41592-019-0686-2.

Vygotsky, L. S. et M. Cole (1978). *Mind in society : Development of higher psychological processes*. Harvard university press.

Wagner, G. P. et L. Altenberg (1996). « Perspective : complex adaptations and the evolution of evolvability ». In : *Evolution* 50.3, p. 967-976.

Wang, C., J. Liu, K. Wu et Z. Wu (2022). « Solving Multitask Optimization Problems With Adaptive Knowledge Transfer via Anomaly Detection ». In : *IEEE Transactions on Evolutionary Computation* 26.2, p. 304-318. DOI : 10.1109/TEVC.2021.3068157.

Wang, J. X., Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran et M. Botvinick (2016). « Learning to reinforcement learn ». In : *arXiv preprint :1611.05763*.

Wang, X., C. Wei, H. Tian, W. Wang et J. Hu (2024). « Implicit predictive behavior cloning for autonomous driving decision-making in urban traffic ». In : *IEEE Transactions on Intelligent Vehicles*.

Watkins, C. J. et P. Dayan (1992). « Q-learning ». In : *Machine learning* 8, p. 279-292.

Wierstra, D., T. Schaul, T. Glasmachers, Y. Sun, J. Peters et J. Schmidhuber (2014). « Natural evolution strategies ». In : *The Journal of Machine Learning Research* 15.1, p. 949-980.

Williams, R. J. (1992). « Simple statistical gradient-following algorithms for connectionist reinforcement learning ». In : *Machine learning* 8, p. 229-256.

Wu, Y., E. Mansimov, R. B. Grosse, S. Liao et J. Ba (2017). « Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation ». In : *Advances in neural information processing systems* 30.

Xia, F., A. R. Zamir, Z. He, A. Sax, J. Malik et S. Savarese (2018). « Gibson Env : real-world perception for embodied agents ». In : *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE.

Xiang, Y., H. Huang, S. Li, M. Li, C. Luo et X. Yang (2023). « Automated Test Suite Generation for Software Product Lines Based on Quality-Diversity Optimization ». In : *ACM Transactions on Software Engineering and Methodology* 33.2, p. 1-52.

Yang, L., Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui et M.-H. Yang (2023). « Diffusion models : A comprehensive survey of methods and applications ». In : *ACM Computing Surveys* 56.4, p. 1-39.

Yang, M. Y., W. Förstner et al. (2010). « Plane detection in point cloud data ». In : *Proceedings of the 2nd int conf on machine control guidance, Bonn*. T. 1, p. 95-104.

Yin, J., A. Zhu, Z. Zhu, Y. Yu et X. Ma (2019). « Multifactorial Evolutionary Algorithm Enhanced with Cross-task Search Direction ». In : *2019 IEEE Congress on Evolutionary Computation (CEC)*, p. 2244-2251. DOI : 10.1109/CEC.2019.8789959.

Yokoya, G., H. Xiao et T. Hatanaka (2019). « Multifactorial optimization using Artificial Bee Colony and its application to Car Structure Design Optimization ». In : *2019 IEEE Congress on Evolutionary Computation (CEC)*, p. 3404-3409. DOI : 10.1109/CEC.2019.8789940.

Zeng, F., W. Gan, Y. Wang, N. Liu et P. S. Yu (2023). « Large language models for robotics : A survey ». In : *arXiv preprint :2311.07226*.

Zhang, M., Y. Lu, Y. Hu, N. Amaitik et Y. Xu (2022). « Dynamic Scheduling Method for Job-Shop Manufacturing Systems by Deep Reinforcement Learning with Proximal Policy Optimization ». In : *Sustainability* 14.9. ISSN : 2071-1050. DOI : 10.3390/su14095177.

Zhang, T., G. Kahn, S. Levine et P. Abbeel (2016). « Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search ». In : *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, p. 528-535.

Zhang, Z. (2012). « Microsoft kinect sensor and its effect ». In : *IEEE multimedia* 19.2, p. 4-10.

Zhao, H., X. Ning, X. Liu, C. Wang et J. Liu (2023). « What makes evolutionary multitask optimization better : A comprehensive survey ». In : *Applied Soft Computing* 145, p. 110545. ISSN : 1568-4946. DOI : https://doi.org/10.1016/j.asoc.2023.110545.

Zhao, T. Z., V. Kumar, S. Levine et C. Finn (2023). « Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware ». In : *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*.

Zhao, W. X., K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong et al. (2023). « A survey of large language models ». In : *arXiv preprint :2303.18223*.

Zhong, J., V. Weistroffer, J.-B. Mouret, F. Colas et P. Maurice (2023). « Workstation Suitability Maps : Generating Ergonomic Behaviors on a Population of Virtual Humans With Multi-Task Optimization ». In : *IEEE Robotics Autom. Lett.* 8.11, p. 7384-7391. DOI : 10.1109/LRA.2023.3318191.

Zhong, J., L. Feng, W. Cai et Y.-S. Ong (2020). « Multifactorial Genetic Programming for Symbolic Regression Problems ». In : *IEEE Transactions on Systems, Man, and Cybernetics : Systems* 50.11, p. 4492-4505. DOI : 10.1109/TSMC.2018.2853719.