Sorbonne Université
Master ANDROIDE

Titre du rapport

UE de projet M1

Paul Cibier – Ziming ZHAO

2025

Table des matières

Chapitre 1

Introduction

Ce rapport présente le projet du Master Androide en Master 1. Nous avons été sous la supervision de Monsieur Stéphane Doncieux (Directeur de l'ISIR) ainsi que de Mademoiselle Mathilde Kappel (Doctorante à l'ISIR).
Pour accéder à notre travail vous pouvez suivre ce lien.

Chapitre 2

État de l'art

Pour ce projet nous basons sur le travail d'une équipe de chercheur de l'ISIR [1], qui utilise des algorithmes Qualité Diversité pour recherchés des points de saisie sur un object.

2.1 Présentation de l'algorithme MAP-Elites

L'algorithme MAP-Elites est principalement divisé en plusieurs étapes : depuis une population initiales, évaluation des individus, mappage de l'espace comportemental, sélection d'individue depuis l'archive pour la mutation, opération de mutation sur ces individue.

Tout d'abord, lors de l'initialisation de la population, un certain nombre d'individus sont générés aléatoirement.
Dans ce projet, le genome est un vecteur de dimension 7, où les trois premières composantes représentent les coordonnées tridimensionnelles de la solution, tandis que les quatre dernières composantes représentent les valeurs du quarternion qui sert à représenter l'object dans l'espace tri-dimensionnel.

Un certain nombres (taille de la population) d'individues par sous-espace comportementale son sélectionner, si l'on parle de la variante Map-Elites sucess alors seulement les meilleurs individues sont sélectionné.

Ensuite, l'algorithme évalue les solutions selon un certain critère (dépendant du scénario et de l'action) pour obtenir leur fitness et leur behavior descriptor. Si un individue est evalué avec success, la valeur de fitness est 1, sinon elle est 0. Pour calculer le behavior descriptor, les valeurs du génome sont tronqué à 3 chiffre après la virgule pour projeter un point spécifique dans une grille, réalisant ainsi la mise en correspondance avec l'espace comportemental.

Enfin, l'algorithme met à jour la population en remappant les individus mutés dans l'espace comportemental. Si un nouvel individu est meilleur que la solution existante, il la remplace. Ces étapes de mutation et d'évaluation sont répétées jusqu'à atteindre le nombre d'itérations prédéfini, et les meilleures solutions dans chaque case de la grille sont sauvegardées.

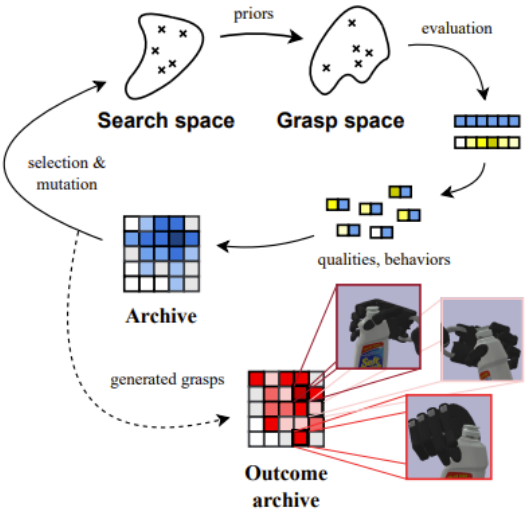

FIGURE 2.1 – MAP-Elites

Chapitre 3

Contribution

3.1 Amélioration des performances et résolution de bug

3.1.1 Outils utilisé

Pour tracker les bottleneck dans le code qu'on a fournit, nous avons utilisé des outils comme perf du noyaux linux qui permet de faire un profilage de l'application en cours d'éxecution et nous affiché les fonction du code les plus executer par notre application. Grace à cela nous pouvons avoir une première intuition sur les parties du code sur lesquelles nous pouvons nous pencher. Pour affiner encore plus notre recherche nous avons utilisé l'outils pySpy qui permet générer des flamesgraph

3.2 Optimisation réalisé

L'optimisation réaliser pour passer du premier flamegraph au deuxième aura été de ne plus fork le processus à chaque évaluation d'un individue mais plutôt de fork n fois le processus uniquement au début de l'éxecution. On fork assez de fois pour avoir autant de processus fils que de cours disponible sur la machine pour crée un "Pool de processus" qui devront attendre de recevoir du travail.
A chaque fois que l'on a besoin de faire évaluer une population les processus vont chercher à accéder à une liste contenant les individues et ils vont les traités, dés qu'un processus fils à fini d'évaluer un individue, il en reprend un autre ect jusqu'a ce que la population est entièrement été évaluer. (Nous avons bien entendue fait attention de bien n'utiliser .imap unordered() en python pour faire en sorte que les processus fils ne s'attende pas les un les autres).

Nous avons également repérer que le robot et l'objet était charger puis supprimer après chaque évaluation. Nous avons corriger ce problème en gardant le robot dans la scène entre les évaluations afin de perdre moins de temps sur des taches redondantes.
Après plusieurs optimisation effectuer d'autre partie du code commence à prendre du temps, comme un mauvais calcul des clé d'un dictionnaire dans l'archive.
Toutes les optimisations mise bout à bout nous a permis de diviser le temps de calcul par 11.
Pour une population de 30 individue et un nombre de génération de 100. Le temps de

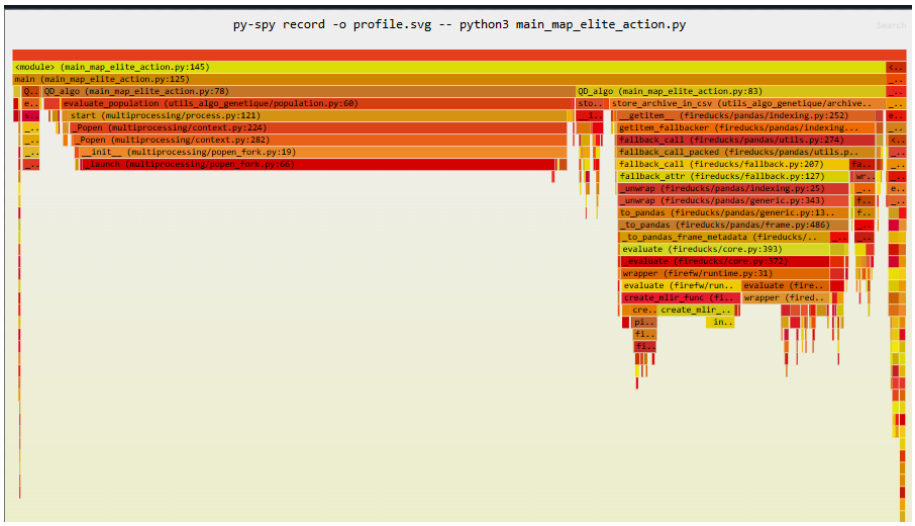

FIGURE 3.1 – On peut voir que l'exécution de pfork prend la majeure Partie de l'éxécution

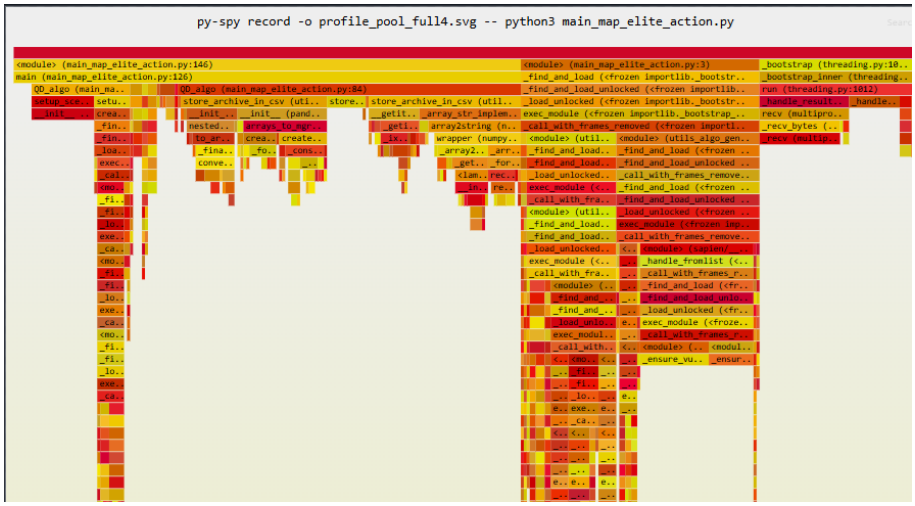

FIGURE 3.2 – On voit maintenant que la partie multiprocessing ne prend qu'une petite partie du temps d'execution et que les fonction lié à l'algorithme QD domine le temps d'execution

calcule initiales était de 165 sec et est maintenant de 15 sec.

3.2.1 Mutation

Après avoir testé l'algorithme MAP-Elites avec la mutation aléatoire, nous devons intégrer les quatre de mutation mentionnés dans le papier [2] dans l'algorithme MAP-Elites, puis comparer et analyser les solutions obtenues sous différents de mutation.

Nous avons testé cinq stratégies de mutation, y compris la mutation random par défaut. Ces stratégies sont les suivantes : **Random Mutation**, **Gaussian Mutation**, **ES Mutation**, **SA Mutation** et **Cov Mutation**.

Nous avons intégré ces stratégies de mutation dans la fonction `mutate_population_from_selction` de la classe `Population` de notre algorithme, ce qui permet de choisir librement la stratégie de mutation en modifiant les paramètres via la fonction `argument_management` dans le fichier principal (`main`).

De plus, nous avons ajouté le nom de la stratégie de mutation utilisée à la fin de chaque fichier de sauvegarde afin de faciliter la distinction et l'analyse des données.

Mutation Random

Le stratégies de mutation par défaut du projet est la mutation aléatoire. Dans cette stratégie de mutation, nous traitons séparément les parties des coordonnées tridimensionnelles et de la posture de rotation du génome.

Pour les trois premières composantes du génome (coordonnées tridimensionnelles), nous générons un nombre aléatoire dans l'intervalle $[0, 1)$ et le multiplions par le coefficient `coefxyz_mutation` afin d'obtenir une variation aléatoire. Cette opération permet d'ajuster légèrement les coordonnées dans une certaine plage.

Pour les quatre dernières composantes du génome (posture de rotation), nous générons également des nombres aléatoires, mais nous les multiplions par un coefficient plus petit `coef_classic_mutattion` afin de garantir que la variation de rotation reste relativement douce.

Mutation Gaussienne

La mutation gaussienne génère de nouvelles solutions en introduisant un bruit aléatoire suivant une distribution normale centrée sur la solution actuelle. L'idée centrale est d'exploiter la propriété de la distribution normale, qui concentre la plupart des valeurs de mutation autour de la valeur actuelle, tout en permettant des mutations plus importantes pour explorer des zones plus éloignées de l'espace de solutions.

Sur le plan mathématique, la mise à jour de la solution avec la mutation gaussienne est donnée par la formule suivante :

$$x' = x + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

Dans cette formule, x représente le vecteur solution actuel et x' est la nouvelle solution après mutation. La variable ϵ correspond au bruit gaussien, généré selon une distribution normale de moyenne nulle et de variance σ^2, où σ est l'amplitude de la mutation.

En ajustant le paramètre σ, on peut donc équilibrer l'exploration locale et globale de l'espace de recherche, rendant la mutation gaussienne particulièrement utile dans de nombreux contextes d'optimisation.

Evolutionary Strategy (ES)

Contrairement à la mutation aléatoire ou gaussienne qui utilisent des paramètres fixes, la mutation ES (Evolutionary Strategy) ajuste dynamiquement l'amplitude de la mutation, permettant ainsi de mieux s'adapter aux variations rencontrées lors de la recherche.

Le principe fondamental de cette approche réside dans la mise à jour auto-adaptative de l'amplitude de mutation. À chaque génération d'évolution, l'amplitude de mutation σ est mise à jour selon la formule suivante :

$$\sigma' = \sigma \cdot \exp(\tau \cdot \mathcal{N}(0, 1))$$

Dans cette formule, σ représente l'amplitude de mutation actuelle et σ' est l'amplitude mise à jour. Le paramètre τ contrôle l'ampleur de la variation et est généralement défini comme $\frac{1}{\sqrt{D}}$, où D est la dimension du génome. $\mathcal{N}(0, 1)$ représente une distribution normale standard avec une moyenne de 0 et une variance de 1. Grâce à ce mécanisme d'auto-adaptation, l'amplitude de la mutation est constamment ajustée pendant l'évolution, permettant à l'algorithme d'effectuer une exploration globale aux premières étapes, puis de converger progressivement vers un optimum local.

Dans notre implémentation, la valeur initiale de l'amplitude de mutation σ_0 est fixée à 0.5 pour favoriser l'exploration globale en début de recherche, tandis que la valeur minimale σ_{min} est définie à 0.025 pour garantir des perturbations fines lors de la phase de convergence.

Simulated Annealing (SA)

La mutation SA est une méthode similaire à la mutation ES, toutes deux visant à ajuster dynamiquement l'amplitude de mutation au cours du processus évolutif. Cependant, contrairement à la mutation ES qui repose sur un ajustement adaptatif, la mutation SA utilise l'idée du recuit simulé. Elle réduit progressivement l'amplitude de mutation au fil des générations, permettant ainsi de bénéficier d'une capacité de recherche globale au début, tout en convergeant progressivement vers une recherche locale plus fine à la fin. La mise à jour de l'amplitude de mutation est donnée par la formule suivante :

$$\sigma' = \text{start} - (\text{start} - \text{stop}) \times \frac{i}{N}$$

Dans cette formule, σ' représente l'amplitude de mutation mise à jour, start est l'amplitude initiale de mutation (fixée à 0.2), stop est l'amplitude finale de mutation (fixée à 0.025), i désigne le nombre d'évaluations déjà réalisées et N est le nombre total d'évaluations

(calculé comme pop_size × nb_generations). Grâce à cette variation linéaire, la mutation SA permet une exploration étendue au début de la recherche, puis une convergence progressive vers des zones plus restreintes et optimisées de l'espace de solutions.

Coverage-based (Cov)

La mutation **Cov** est similaire à la mutation **SA**, mais son ajustement d'amplitude ne dépend pas de l'avancement du processus d'évolution, mais plutôt du taux de couverture de l'espace des solutions.

L'amplitude de mutation σ est calculée selon la formule suivante :

$$\sigma = \text{start} - (\text{start} - \text{stop}) \times \text{Coverage}$$

Ici, start et stop représentent respectivement l'amplitude de mutation initiale et minimale, tandis que Coverage désigne le taux de couverture actuel. Grâce à cette approche, l'algorithme peut effectuer une exploration globale lors des premières étapes, lorsque la couverture est faible, puis réduire progressivement l'amplitude pour effectuer une recherche locale à mesure que la couverture augmente.

Dans notre implémentation,, nous avons défini la valeur initiale (start) à 0.4 et la valeur minimale (stop) à 0.025.

3.2.2 Analyse

Diversité des solutions : couverture et entropie

Pour évaluer la diversité d'une stratégie de mutation, nous n'avons pas seulement utilisé le score de couverture requis, mais nous avons également introduit l'entropie comme référence supplémentaire.

Le score de couverture est calculé comme le rapport entre le nombre de cellules contenant des solutions et le nombre total de cellules. Dans ce projet, comme chaque cellule ne conserve que la meilleure solution, le nombre de cellules contenant des solutions correspond au nombre de descripteurs comportementaux (*behavior descriptor*). Cette valeur peut être utilisée pour mesurer si l'algorithme explore suffisamment d'espaces et produit des solutions diversifiées.

Cependant, nous estimons que le score de couverture ne suffit pas pour représenter la diversité dans ce projet. Comme nos expériences portent sur des objets réels, nous souhaitons explorer un maximum de positions pour réaliser une action donnée. Par exemple, lors de la tentative de saisie d'une boîte, nous espérons pouvoir la saisir sous différents angles. Or, le score de couverture ne permet pas d'évaluer la répartition des solutions.

C'est pourquoi nous avons décidé d'introduire l'entropie comme référence supplémentaire. La formule de l'entropie est la suivante :

$$H = -\sum_{i=1}^{n} p_i \log(p_i)$$

où :

— p_i : la probabilité de la i-ème sous-région, avec $\sum p_i = 1$,
— n : le nombre total de sous-régions.

Ici, nous avons divisé l'espace des solutions en $10 \times 10 \times 10$ sous-espaces, puis calculé la probabilité d'apparition des solutions dans chaque sous-espace. Enfin, nous avons appliqué la formule de l'entropie pour obtenir sa valeur. Plus la valeur de l'entropie est élevée, plus la distribution des solutions est uniforme et le niveau de désordre est élevé. Dans le cas extrême où toutes les solutions se concentrent dans un seul sous-espace, la valeur de l'entropie sera nulle ($\log 1 = 0$).

En raison de l'imperfection du simulateur Genesis, nous avons utilisé le simulateur Sapien pour garantir la rigueur des données. Cependant, Sapien ne prend pas en charge le mode `GPU_parallel`, et le mode `GPU_simple` nécessite un temps de calcul trop long. Par conséquent, nous avons exécuté chaque stratégie de mutation trois fois, puis nous avons pris la médiane du score de couverture pour représenter la performance de cette stratégie de mutation.

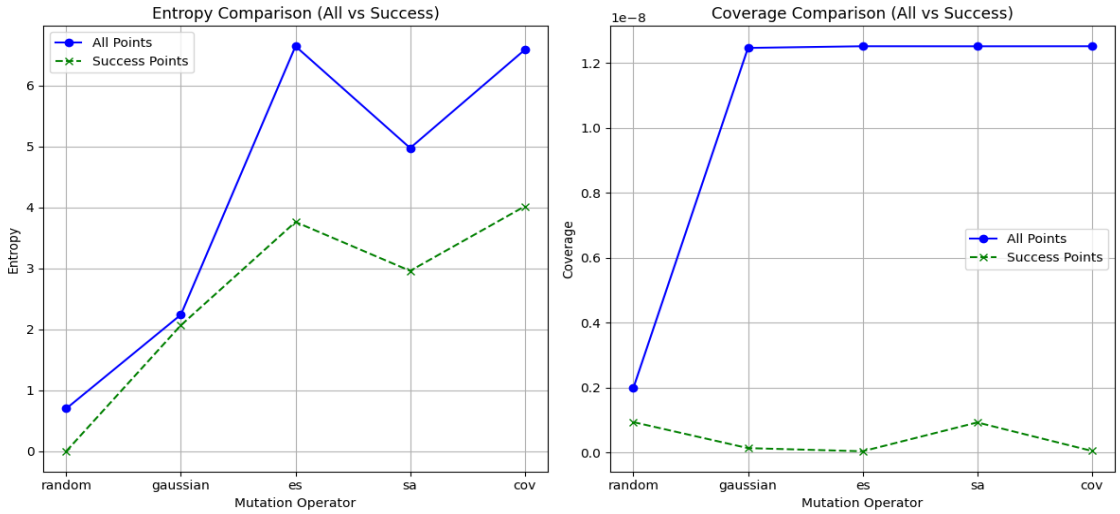

FIGURE 3.3 – Entropy et Coverage comparison

D'après les résultats, nous pouvons constater que les stratégies de mutation dynamique présentent un score de couverture nettement supérieur aux stratégies classiques telles que la mutation gaussienne et la mutation aléatoire. Cependant, la différence de score de couverture entre les trois stratégies de mutation dynamique n'est pas significative.

En revanche, les valeurs d'entropie ne montrent pas la même tendance. Par exemple, pour les stratégies *cov* et *sa*, lors du calcul du score de couverture des solutions réussies, *sa* est supérieur à *cov*, ce qui signifie que la diversité de *sa* est plus élevée que celle de *cov*. Cependant, les valeurs d'entropie révèlent une relation inverse : l'entropie de *sa* est inférieure à celle de *cov*.

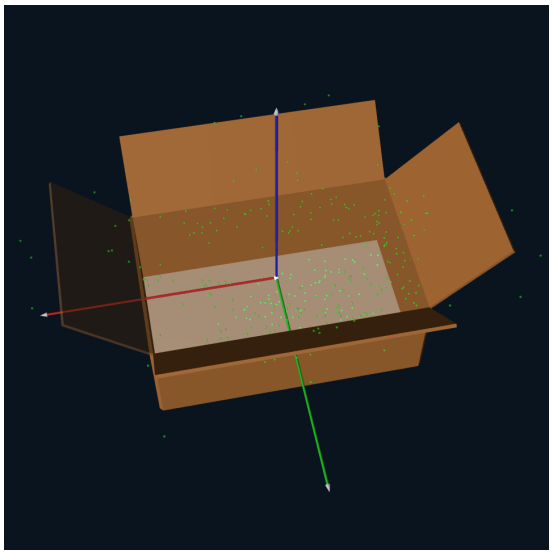

FIGURE 3.4 – Sucess point Cov

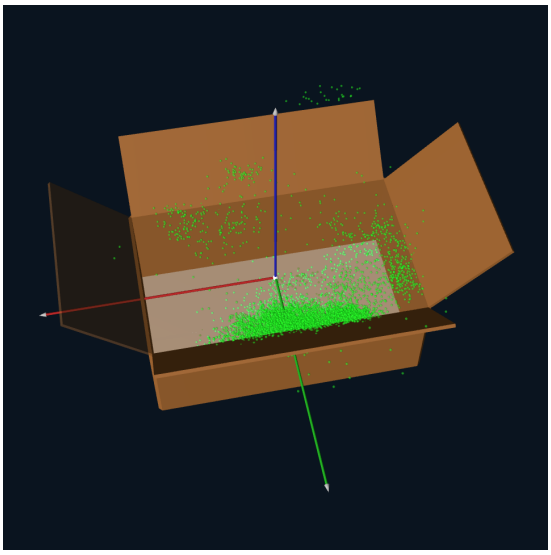

FIGURE 3.5 – Sucess point SA

Comme le montre la figure, après avoir affiché les solutions réussies, nous pouvons constater que bien que les solutions générées par SA soient plus nombreuses que celles de Cov, ces dernières sont plus dispersées. En d'autres termes, Cov explore des solutions plus éloignées dans l'espace de recherche.

3.3 Map dynamique et adaptative

3.3.1 Problématique

Lors de la recherche de solution pendant l'exécution d'algorithme QD. L'algorithme ne sait pas où sont les points de success. Donc au début il génère une population initiale, lors de l'évaluation de cette population initiale beaucoup d'entre eux seront des individues sans success, ceux sans success sont insérer dans la map dans leurs sous espace correspondant si aucun autre individue ne s'y trouvait, pour les individues à success ils sont placé dans la map si aucun individue ne s'y trouvait ou qu'ils sont meilleur que l'individue déjà présent dans le sous espace leurs correspondant. En suite l'algorithme sélectionne des individues depuis l'archive de manière aléatoire. Recommence ce processus avec comme population composé des individues sélectionner précédemment. et recommence ce processus. Lors de nos entretiens avec nos responsable de projet, ils nous ont expliqué que leurs buts était de générer une assez grande quantité de point et dans l'exemple d'un micro onde avec l'action de saisie, il n'y a que la poigné du micro-onde qui est intéressante et qui générera des succès.

Le problème étant que si ils veullents beaucoup de point alors ils doivent augmenter la discrétisation de l'espace de la grille de l'agloritme QD. Pour faire muter le plus de point possible prêt de la poigné. Le problème avec l'augmentation de la discrétisation est qu'on augmente la discrétisation dans des zones qui ne nous intéresse pas forcément. Et donc il s'était aperçue que l'algorithme passait beaucoup de temps à explorer des zones qui n'était très intéressante du fait de la forte discrétisation.

3.3.2 Solution Proposé

Les chercheurs Stéphane Doncieux et Mathilde Kappel nous ont donc exposé une idée lors de notre entretiens. Sera t il possible que l'algorithme converge plus rapidement pour générer des points de success si l'on pouvait augmenter la discrétisation de la map dans les zones à success en partant du discrétisation de l'environnement faible jusqu'a arriver à une discrétisation forte dans les zone à fort succes.

3.3.3 Mise en oeuvre

Structure de donnée et algorithme

Pour ce faire nous avons choisie d'implémenter cette idée à l'aide d'un arbre. Les feuilles représente un sous espace contenant un individu. et les noeuds de l'arbre représente un sous espaces de la map qui avait besoin d'être discrétiser encore plus.

Pour savoir dans quelle sous espace un point doit se situé, on doit parcourir chaque dimension du behaviour descriptor et pour chaque dimensions qui sera supérieur à la valeur du centre du sous espace, on met le bit à 1 de l'index à la position du numéro de la dimensions testé.

exemple : Si la deuxième dimensions est supérieur a la deuxième dimensions du centre du sous espace mais que les autres sont en dessous, seul le deuxième bit de l'index vaudra 1. et les autres 0. donc l'index de ce point vrauda 2. Cela évite de tester tout les nouveau sous espace pour savoir si l'on ce trouve à l'intérieur ou pas. Pour rappel si l'on a 32 dimensions dans le behavior descriptor alors chaque dimensions est coupé en deux lorsque l'on discrétise plus. Ce qu'il nous fait 2 puissance 32 sous espace. Ce qui fait beaucoup de dimenssions à essayer. Avec ma méthode on y arrive de manière linéaire par rapport à la dimensions et non de manière exponentiel comme le voudrais l'approche naïve.

Choix du language

Pour le choix du language de programmation, nous avons choisie de nous orienter vers le Rust qui nous offre un très bon contrôle sur la manipulation des pointeurs pour l'arbre ainsi qu'une garantie que la gestion de la mémoire ne produise pas de fuite de mémoire ou de bug lié à la gestion de celle-ci. En effet ces algorithmes sont destinées à s'exécuter pendant très longtemps Il serait catastrophique de faire planté le programme qui s'execute à cause d'un Out-Of-Memory error produit car nous aurions mal libéré la mémoire par exemple avec un code en c/c++.

3.3.4 Resultat

Pour faire nos test nous avons executer l'algorithme avec le simulateur Sapien car la version avec Génésis est encore en cours de developpement. Seulement 1 execution à pu être générer en raison de la difficulté de simulation, nous avons manquer de temps de calcul qui peuvent prendre énormement de temps.

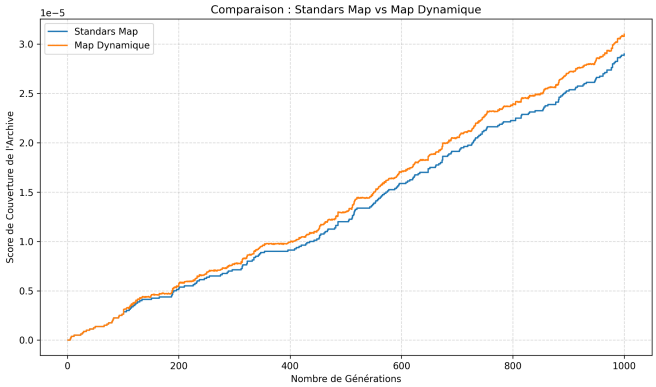

FIGURE 3.6 – Score de converture pour les différentes méthodes d'archivage des solutions

On peut voir que au début les deux algorithmes converge à la même vitesse mais à partir de 50 génération environ la version avec la map Dynamique commence à converger légèrement plus vite que la version de l'archive standard.

Chapitre 4

Conclusion

4.1 Operateur de mutation - Conclusion

Dans ce projet, nous avons exploré l'impact de différents opérateurs de mutation sur l'algorithme MAP-Elites. À travers des expérimentations systématiques, nous avons comparé cinq opérateurs de mutation : Random, Gaussian, ES, SA et Cov.

Les résultats montrent que les opérateurs de mutation dynamiques (ES, SA, Cov) sont nettement supérieurs aux méthodes classiques de mutation aléatoire et gaussienne en termes de couverture. Ces stratégies dynamiques permettent d'adapter la portée des mutations en fonction de l'avancement de l'évolution ou de la couverture actuelle, ce qui favorise à la fois l'exploration globale et la convergence locale.

Cependant, la seule couverture ne suffit pas à évaluer la diversité des solutions générées. C'est pourquoi nous avons également introduit l'entropie comme indicateur complémentaire. L'entropie permet de mesurer la répartition des solutions dans l'espace de recherche, ce qui est particulièrement pertinent pour évaluer la diversité. Par exemple, bien que l'opérateur SA ait obtenu une couverture plus élevée que Cov dans certains cas, son entropie était inférieure, indiquant une répartition moins homogène des solutions.

Ainsi, l'intégration des opérateurs de mutation dynamiques et de l'entropie dans l'évaluation a permis d'améliorer significativement la qualité et la diversité des solutions générées par l'algorithme MAP-Elites.

4.2 Map Dynamique - Conclusion

Nous avons pu observer que la map Dynamique représente un léger avantage au début du l'expérience et permet de sélectionner plus d'individue à success mais uniquement dans la version standard de Map-Elites. Pour Map Elites sucess, notre stratégie revient à prendre tous les points de sucess ce qu'il ne garantie pas la diversité. Pour ces raisons nous avons décider de n'évaluer que la version standard de MAP-Elites pour les tests avec la map dynamique.

Bibliographie

[1] Johann HUBER, François HÉLÉNON, Mathilde KAPPEL, Elie CHELLY, Mahdi KHORAMSHAHI, Faïz Ben AMAR et Stéphane DONCIEUX. *Speeding up 6-DoF Grasp Sampling with Quality-Diversity*. 2024. arXiv : 2403.06173 [cs.RO]. URL : https://arxiv.org/abs/2403.06173 (page 2).

[2] Jorgen NORDMOEN, Eivind SAMUELSEN, Kai Olav ELLEFSEN et Kyrre GLETTE. *Dynamic mutation in MAP-Elites for robotic repertoire generation*. 2018. URL : https://direct.mit.edu/isal/proceedings-pdf/alife2018/30/598/1904764/isal_a_00110.pdf (page 6).

Annexe A

Cahier des charges

Pour ce projet nous avions la charge de trouver une manière de contribuer à la thèse d'une doctorante (Mathilde Kappel). Nous avions ainsi la charge de contribuer à son code afin de proposer des amélioration.
Nous avions la charge d'implémenter de tester différents opérateur de mutation.
Ainsi nous avions également la charge d'implémenter et d'étudier une map dynamique qui aurait le potentiel d'accélérer la convergence du score de converture de success.

Annexe B

Manuel utilisateur

Pour utiliser ce projet il est necessaire de suivre les readmes de chaque sous projet. https ://github.com/Paulo-21/Projet-ANDRO-24-25