Université Pierre et Marie Curie,
ISIR,
École Normale Supérieure

Caractérisation de comportement dynamique en robotique mobile & application de la robotique évolutionniste

Stage court du Master 2 Concepts Fondamentaux de la Physique

Claude Perdigou
claude.perdigou@ens.fr

Sous la direction de
Stéphane Doncieux

10 janvier - 07 mars 2011

Table des matières

Introduction

Mon stage du Master 2 Concepts Fondamentaux de la Physique d'une durée de deux mois a eu lieu dans les locaux de l'ISIR, laboratoire de l'université Pierre et Marie Curie. Il m'a permis de m'initier à un domaine qui m'a toujours attiré, et grâce à l'encadrement sérieux et à l'expérience des membres du laboratoire, j'ai beaucoup appris malgré la brièveté de mon séjour parmi eux. De plus, j'ai aidé un des projets grâce à mes connaissances en physique statistique, et contribué à un second.

Les problématiques et les méthodes du laboratoire sont majoritairement issues du domaine de l'informatique, mais les problèmes étudiés ont très souvent une importante dimension relevant de la physique. La façon dont le laboratoire traite des problèmes de robotique m'a beaucoup attiré. J'y ai découvert l'approche Animat et les algorithmes évolutionnistes avec beaucoup d'intêret, et j'en retiens une expérience extrêmement enrichissante. Ce stage m'a permis de compléter ma formation en acquérant des compétences en informatique et en programmation, et en apprenant à maîtriser une méthode qui pourrait être appliquée de façon fructueuse à des problèmes de physique.

Ce rapport est constitué de trois parties. La première sert de longue introduction et permettra de bien préciser le domaine et le contexte auxquels se rattache le problème étudié. La seconde et la troisième parties de ce rapport présentent les résultats obtenus dans les deux sujets sur lesquels j'ai travaillé durant mon stage.

1 Les algorithmes évolutionnistes et leur application à la robotique

1.1 Algorithmes évolutionnistes

Les algorithmes évolutionnistes (ou évolutionnaires) désignent un ensemble d'algorithmes stochastiques qui utilisent les principes de l'évolution naturelle pour résoudre des problèmes divers, très majoritairement d'optimisation. Leur fonctionnement repose sur la génération aléatoire de solutions potentielles, puis la sélection des meilleurs candidats selon un critère préétabli. Les individus sélectionnés serviront de parents à la génération ultérieure et les mutations/croisements seront réalisés à partir du génotype de ceux-ci. Ce processus est alors répété sur de nombreuses générations, et grâce à la pression de sélection ainsi générée, l'algorithme converge vers une solution efficace au problème posé.

Le fonctionnement de ces algorithmes peut être décrit plus formellement par la procédure suivante :

- Initialisation aléatoire d'une population de départ
- Évaluation de toute la population selon un ou plusieurs critères (objectifs)
- **Boucle** jusqu'à un 'critère de fin' :
  - Sélection des meilleurs individus
  - Mutation/croisement des *individus sélectionnés* pour recréer une population complète
  - Évaluation de toute la population selon les objectifs

La réussite d'une telle procédure repose évidemment sur plusieurs prérequis non négligeables :

- La capacité de trouver des critères de sélection pertinents
- La capacité d'évaluer en un temps raisonnable toute la population

Même dans le cas où ces conditions sont réunies, pour certains problèmes "difficiles", aussi appelés "trompeurs", l'algorithme peut très bien ne jamais converger vers une solution satisfaisante. Soit parce que celle-ci est trop difficile à atteindre (temps de convergence infini), soit parce que l'algorithme converge prématurément vers une solution *localement optimale*, et ne peut plus sortir de ce "cul-de-sac évolutionniste".

Pour pallier à ce problème, plusieurs idées ont été implémentées par le passé. L'une de celle-ci, l'introduction d'un *objectif de diversité*, est décrite au paragraphe 1.3 et constitue l'un des thèmes de mon stage.

On distingue trois grands types d'algorithmes évolutionnistes :

- Les algorithmes génétiques :
  Ce sont les algorithmes évolutionnistes les plus usuels. Les génotypes évolués sont des vecteurs de valeur (souvent binaires), qui correspondent à un phénotype et dont on évalue la capacité à résoudre un problème. A chaque génération, tous les parents disparaissent et laissent place à une population totalement composée de leurs descendants.
- Les stratégies d'évolution :
  Elles sont très similaires à un algorithme génétique, mais cette fois la population est recréée à partir des parents et des enfants. Autrement dit, la sélection se base sur un classement regroupant parents et enfants. Ces derniers ne sont sélectionnés que s'ils sont mieux adaptés que leurs parents.
- La programmation évolutionnaire :
  Les objets étudiés sont des programmes informatiques, qu'on évalue sur leur capacité à résoudre une tâche informatique.

1.2 Application à la robotique

1.2.1 Introduction

L'application de telles méthodes à des problèmes de robotique a commencé très récemment, en 1992, dans le but de proposer une approche différente de celle jusqu'ici utilisée. Ces méthodes ouvriront la voie à l'approche Animat, qui prétend faire de la robotique en s'inspirant de la nature. Les travaux sur l'intelligence artificielle ont longuement été dominés par une approche de type "ingénieur" : le robot est une machine qui est programmée pour effectuer des tâches. Cette approche présente plusieurs problèmes difficiles à surmonter. Premièrement, les capacités du robot sont limitées par l'agilité des programmateurs à lui inculquer la façon de réaliser les tâches désirées. Deuxièmement, un tel robot ne peut faire preuve d'"imagination", sauf si elle lui a été programmée (elle est donc limitée). En donnant la liberté au robot de se créer sa propre représentation du monde, on lui laisse la possibilité de développer à la fois des capacités inattendues et des réactions non programmées.

Bien entendu, concevoir un robot à l'aide d'un algorithme évolutionniste demande beaucoup de puissance de calcul, d'autant plus si la tâche est difficile et longue à évaluer. C'est aussi pour cette raison que les applications des algorithmes évolutionnistes au domaine de la robotique sont récentes puisqu'on dispose de plus en plus de ces capacités de calcul.

En pratique, au lieu de programmer les degrés de liberté d'un robot, on les transforme en paramètres d'un algorithme évolutionniste qui va simuler le comportement de chaque individu et l'évaluer sur une tâche. Un exemple simple est celui d'un robot marcheur : la fitness est la distance parcourue et on donne à l'algorithme les caractéristiques des mouvements imposés aux pattes (fréquence, amplitude, et déphasages de sinusoïdes

par exemple). D'autres exemples plus complexes peuvent être des robots volants, des robots qui doivent se repérer dans un labyrinthe, etc.

1.2.2 Aspects techniques

Pour faire fonctionner un algorithme évolutionniste sur une expérience précise, il faut disposer d'un espace de travail qui permette de faire correspondre l'algorithme lui-même, avec la partie analyse (mesure des différentes grandeurs), la partie simulation (mesure de la fitness des individus) et la partie statistique (exportation de données). Tous ces morceaux sont des programmes a priori distincts en C++, mais ils communiquent grâce au framework sferes2, développé à l'ISIR [1] il y a quelques années. Les parties d'analyse et de statisique ne sont souvent que du traitement simple de données et de l'exportation, ce qui peut être codé directement dans l'expérience. Par contre, les parties simulation et algorithme évolutionniste sont des morceaux de code indépendants et conséquents, qui sont appelés par le framework à chaque étape.

1. La partie évolution : L'algorithme évolutionniste le plus utilisé au laboratoire est NSGA-II, il est développé par K. Deb et son équipe [2]. C'est un algorithme multiobjectif, ce qui le rend utilisable sur des problèmes complexes où un compromis doit être atteint. Il est de plus très rapide, grâce à une méthode de classement des individus innovante. Pour le lecteur intéressé, cette procédure très technique est détaillée dans l'article précédemment cité.

   Une partie importante, qu'il est impératif de détailler concerne la sélection des individus dans le cadre des algorithmes multiobjectifs. En effet, les individus obtiennent des classements selon chacune des fitness choisies, et l'on obtient des populations réparties de la façon suivante sur un graphe en fitness :

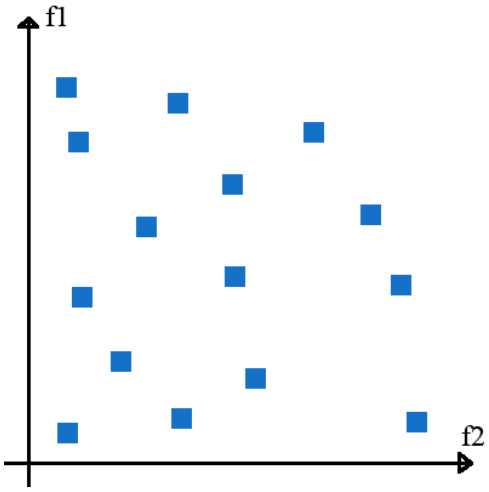

FIG. 1 – Individus d'une population sur un graphe de fitness.

Pour sélectionner les meilleurs individus, on introduit la notion de domination. On dit qu'un individu en domine un autre s'il possède un score supérieur ou égal sur toutes les fitness, et strictement supérieur sur au moins une des fitness.
En pratique, cette notion permet de définir le front de Pareto : c'est l'ensemble des individus qui ne sont pas dominés. Le front de Pareto de l'image précedente est le suivant :

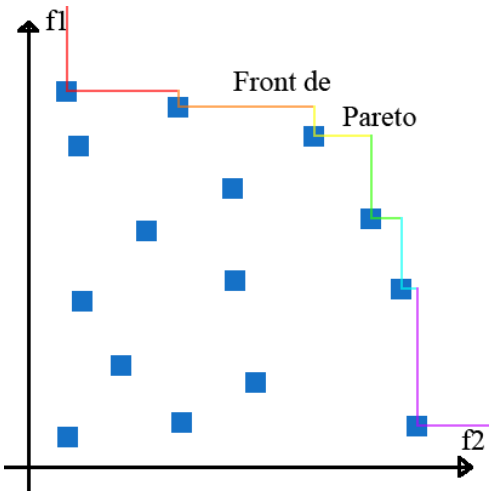

FIG. 2 – Le front de Pareto d'une population d'individus.

On remarque bien que chaque point appartenant au front coloré n'est pas dominé. On remarque aussi que cette définition équivaut à celle-ci : le quart de plan nord-est d'un individu sur le front de Pareto est inoccupé.

L'algorithme évolutionniste sélectionnera les individus sur le front de Pareto (puis, s'il n'y en a pas assez, sur le second front de Pareto, puis le troisième, etc...). NSGA-II fait cela de façon pertinente, car il mesure aussi la densité d'individus locale sur le front de Pareto. Supposons que beaucoup d'individus très proches se trouvent sur le front, en sélectionnant des individus au hasard on conservera cette même distribution en densité. NSGA-II pondère la probabilité de sélectionner un individu par son degré d'isolement. Ainsi on garde une plus grande diversité à la génération suivante :

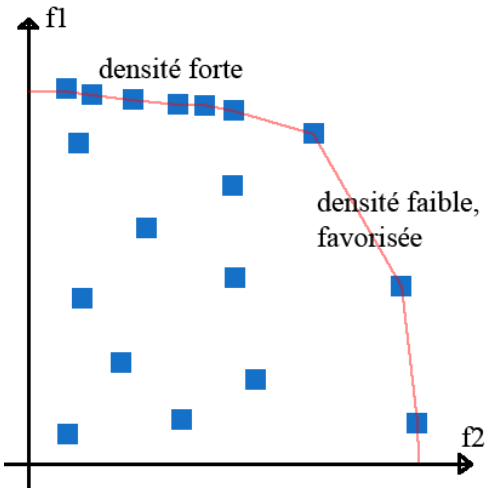

FIG. 3 – Le système d'handicap qui permet de garder une densité constante sur le front de Pareto.

2. La partie simulation : Pour évaluer des robots sur des tâches ayant lieu dans un environnement physique, il faut disposer d'un simulateur de ce problème. Ce simulateur doit impérativement être très rapide, car la simulation est exécutée à chaque génération pour tous les individus. Fatalement, un tel simulateur sera peu précis... Ceci soulève le problème de la transférabilité, le fait qu'un comportement simulé puisse ne pas du tout fonctionner sur un robot réel. C'est un problème difficile, qui est aussi étudié à l'ISIR. De façon très intéressante, le thésard travaillant sur le sujet est parvenu à établir un critère de transférabilité, qu'il suffit d'ajouter aux objectifs d'évolution pour sélectionner les solutions transférables ! [3]

   Le simulateur utilisé se nomme ODE (pour Open Dynamics Engine). Ce n'est pas le seul qui puisse être utilisé (Bullet par exemple est une alternative), mais sa simplicité et la fiabilité que lui ont fait gagner plusieurs années de débugage en font un outil simple et fonctionnel. La procédure consiste à modéliser l'environnement du robot comme un paysage fixe dans le simulateur, puis à modéliser le robot comme objet mobile pouvant entrer en collision avec cet environnement. Enfin on place sur la structure du robot les moteurs et les servos qui servent à son déplacement. Une fois la boucle de collision mise en place, la simulation est prête !

   Je décrirai de nouveau cette procédure au paragraphe 3.2 avec un exemple concret.

1.3 Importance de la diversité

Comme nous l'avons souligné précédemment, certains problèmes "trompeurs" peuvent être particulièrement difficiles pour un algorithme évolutionniste. Le phénomène se comprend aisément si l'on visualise un labyrinthe de la forme suivante :

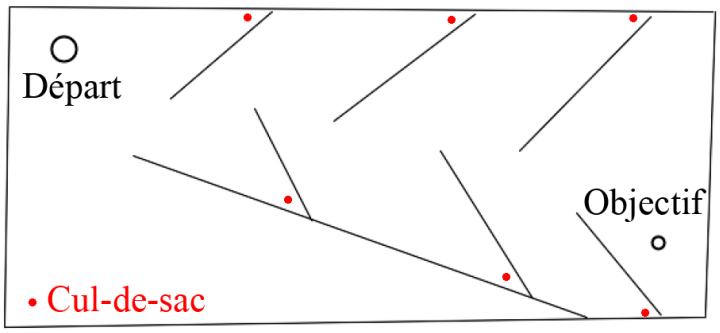

FIG. 4 – Une arène trompeuse : les robots doivent se déplacer du départ à l'arrivée.

Dans ce problème, la population de robots va rapidement se diriger vers son objectif...pour se retrouver coincée dans un cul de sac. Il y a peu de chances qu'un individu ressorte de ce cul-de-sac : il faudrait une mutation suffisamment forte pour créer un robot qui s'échappe de ce trou et parvienne ainsi à s'approcher davantage de son objectif. Dès lors, c'est ce robot qui sera selectionné, et le même problème se présente de nouveau avec le cul de sac suivant. Plusieurs approches existent pour forcer les robots à avoir des comportements différents, à innover.

La première méthode consiste à sélectionner les robots sur un critère de nouveauté seulement. [4]

La deuxième méthode, majoritairement développée à l'ISIR, consiste à optimiser un critère de diversité, en même temps que la fitness. Ceci est possible en utilisant un algorithme évolutionniste multi-objectifs.

2 Caractérisation d'un comportement dynamique

2.1 Présentation du robot étudié

Mon travail sur les critères de diversité a été appliqué à une expérience particulière qui est décrite ici. La tâche consiste à ramasser des balles dans une arène, qu'il faut ensuite acheminer jusqu'à un panier où elles doivent être déposées. L'arène est de plus munie d'un interrupteur, qui ouvre une porte menant à une salle où quatre balles supplémentaires se trouvent. L'arène a la forme suivante :

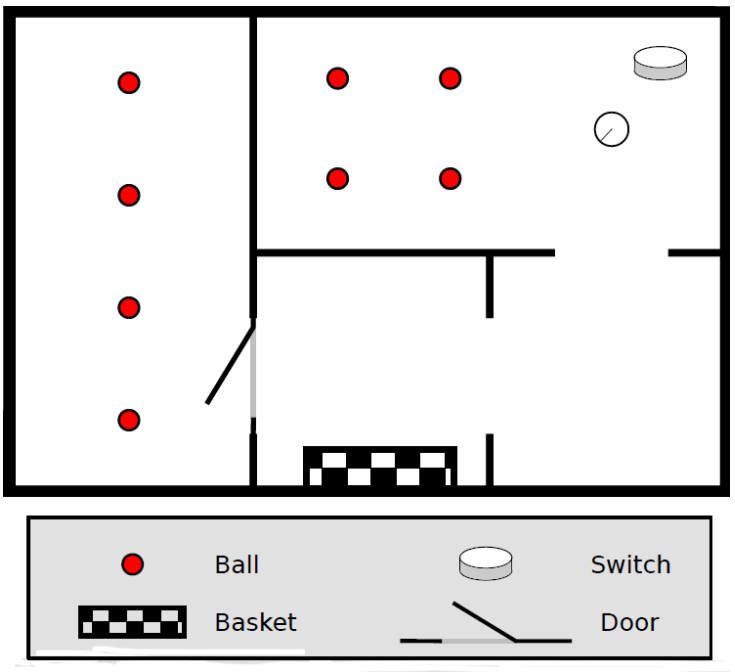

FIG. 5 – L'arène pour l'expérience du ramassage de balles.

La fitness est tout simplement le nombre de balles ramassées. L'algorithme fait évoluer le réseau de neurones qui commande le robot. Il est composé au départ de 15 neurones, correspondant aux 15 capteurs et effecteurs du robot. Les mutations possibles sont : l'apparition d'un neurone caché, la disparition d'un de ceux-ci, l'apparition d'un lien entre deux neurones, la disparition d'un de ceux-ci, et enfin la modification du poids d'un de ces liens pris au hasard. Ces mutations ont lieu avec une probabilité fixe, le taux de mutation associé à chacune. À titre d'exemple, un réseau de neurones typique obtenu à la fin du processus d'évolution ressemble à ceci :

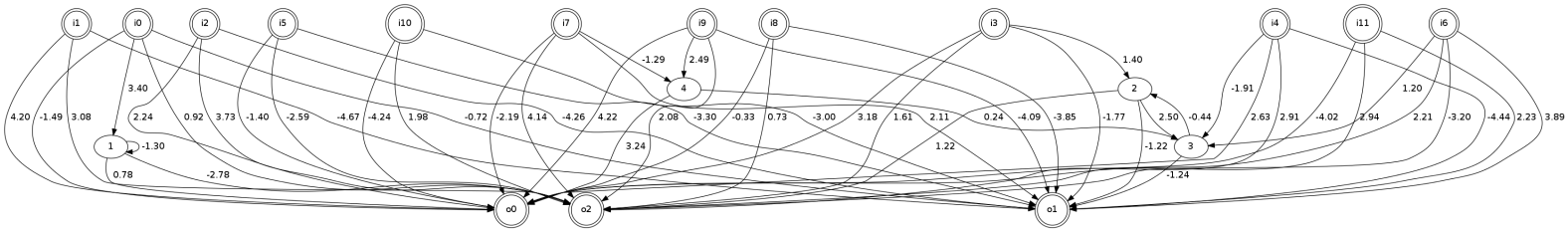

FIG. 6 – Le réseau de neurones d'un invididu obtenu par évolution.

La difficulté de cette tâche a été volontairement augmentée, afin de voir dans quelle mesure un algorithme évolutionniste peut trouver des solutions à un problème très difficile. Ainsi, pour avoir la fitness maximale, il faut avoir accès à toutes les balles, et donc avoir activé l'interrupteur. Or rien ne récompense le robot s'il active l'interrupteur, il doit

donc naturellement avoir un comportement d'exploration important. Deuxièmement, si le robot lâche une balle sur le sol, mais pas dans le panier, cette balle disparaît de l'arène et est définitivement perdue. Enfin, la forme de la fitness elle-même rend la tâche difficile. En effet, le robot doit passer par de nombreuses étapes pour gagner ne serait-ce qu'un seul point de fitness : trouver une balle, la ramasser, la garder avec lui et aller jusqu'au panier, puis enfin lâcher la balle. Ce sont beaucoup d'étapes non guidées, et le robot doit développer un comportement riche pour pouvoir les accomplir.

Ce problème est suffisamment difficile pour qu'un run lancé uniquement en maximisant la fonction de fitness mène toujours à des résultats nuls. L'espace des paramètres à explorer est tellement immense que l'algorithme ne convergera jamais ! C'est un exemple pratique où un critère de diversité est absolument nécessaire. En effet, lorsqu'on maximise deux paramètres, la fonction de fitness *et* une mesure de diversité, l'algorithme parvient à générer des solutions au problème ! Nous verrons ci-après ce qu'est un bon critère de diversité, et ceux que l'on peut utiliser.

2.2 Un critère de diversité généraliste

L'importance des critères de diversité pour assurer la convergence d'un algorithme évolutionniste sur des problèmes "trompeurs" donne un grand intérêt à la conception de nouveaux critères.

Un critère de diversité idéal doit être le meilleur pour un nombre maximal de problèmes différents, il doit donc être généralisable. En effet, on peut aisément se persuader que pour chaque problème étudié, on saura trouver un critère de diversité suffisamment efficace pour faire converger l'algorithme. Par exemple, sur un problème d'exploration, on peut comparer les trajectoires. Mais ces critères spécifiques ne sont pas généraux, alors qu'on aimerait pouvoir extraire ces informations, i.e. le comportement du robot, à partir de données d'un niveau inférieur. En y réfléchissant, le seul point commun entre tous les robots est qu'ils possèdent des capteurs et des effecteurs, qui sont des composants électroniques. Ces composants renvoient des voltages au cours du temps, et il est possible de récupérer cette chaîne de nombres flottants en fonction du temps. Ces données sont faciles à traiter, et on veut en extraire des informations *physiques*. Ceci devrait être possible, puisque ces données de commande comportent la totalité de l'information : en les connaissant on peut retrouver le comportement du robot (actions, trajectoire). Un critère de diversité capable d'exploiter directement ces données a immédiatement deux avantages : c'est un critère général (applicable à n'importe quel robot sur n'importe quel problème) et il est très rapide à évaluer (traitement de chaînes de caractères).

Plusieurs méthodes sont envisageables, et il existe déjà une bibliographie assez riche sur le sujet.[5] La première idée qui vient à l'esprit d'un physicien lorsqu'on évoque les mots "information" et "chaîne de caractères" dans la même phrase, est évidemment l'entropie, au sens de la théorie de l'information. Il s'avère que cette idée est elle aussi documentée,[6, 7] mais n'avait jamais été implémentée à l'ISIR. J'ai donc aidé à la mise

en place d'un comparatif de l'efficacité des différents critères de diversité utilisés par le laboratoire. Il a fallu commencer par évaluer dans quelle mesure l'entropie constituait un critère de diversité convenable.

Le critère Entropie

Le robot possède deux types de capteurs, les capteurs binaires et les capteurs flottants. Les premiers ne peuvent prendre que deux valeurs : 0 ou 1. Par exemple, le capteur qui détecte la présence du panier est un de ceux-ci. Il vaut 1 si le panier est en vue, 0 sinon.
Les seconds peuvent prendre n'importe quelle valeur comprise entre 0 et 1. Le capteur de distance aux murs est l'un de ceux-ci. Il vaut 1 lorsque l'on est à plus d'une distance limite d'un mur, puis décroît linéairement jusqu'à atteindre la valeur 0 au contact du mur. Pour ce deuxième type de capteurs, il est nécessaire de discrétiser leurs valeurs pour pouvoir calculer leur entropie. On choisit de diviser l'intervalle $[0, 1]$ en dix intervalles de tailles égales, $[0.1 * i, 0.1 * i + 1[$. La valeur 1 est ici et dans la suite ajoutée au dernier intervalle. p_i, la probabilité qu'un capteur ait une valeur comprise dans l'intervalle précédent est évaluée ainsi :

Pour $t \in [[1, N]]$:

$$i = floor[10 * V(t)] + 1$$
$$p_i \mathrel{+}= 1/N$$

$V(t)$ est la valeur renvoyée par le capteur en question au temps $t \in [[1, N]]$

Une fois ces probabilités p_i établies pour les capteurs binaires et flottants, leurs entropies se calculent selon les formules suivantes :

Capteurs binaires : ils prennent les valeurs 0 ou 1, donc

$$S_{capteur\ binaire} = -\sum_{i=0,1} p_i \frac{ln(p_i)}{ln(2)}$$

p_0 et p_1 sont les probabilités que le capteur ait la valeur 0 ou 1 respectivement.

Capteurs flottants : ils prennent leur valeur entre 0 et 1, donc

$$S_{capteur\ flottant} = -\sum_{i=1..10} p_i \frac{ln(p_i)}{ln(10)}$$

p_i est la probabilité décrite précédemment.

Un échantillon de 10 comportements différents m'a été soumis (choisis de façon adéquate par une autre personne), et j'ai dû en aveugle essayer de retrouver lesquels avaient

des comportements semblables, et lesquels étaient très différents. Le tableau d'entropie obtenu est le suivant :

Capteur \ Individu	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.86	0.87	0.90	0.94	0.85	0.90	0.73	0.88	0.85	0.75
2	0.81	0.87	0.95	0.86	0.94	0.73	0.84	0.85	0.62	0.64
3	0.81	0.90	0.95	0.85	0.92	0.85	0.80	0.92	0.88	0.83
4	0.87	0.66	0.76	0.87	0.83	0.84	0.93	0.73	0.22	0.46
5	0.96	0.63	0.93	0.99	0.99	0.96	0.54	0.76	0.48	0.73
6	0.19	0.71	0.71	0.85	0.51	0.50	0.63	0.61	0.99	0.15
7	0.22	0.85	0.73	0.61	0.84	0.85	0.99	0.48	0.85	0.22
8	0.16	0.03	0.14	0.61	0.15	0.32	0.51	0.05	0.49	0.20
9	0.16	0.16	0.22	0.08	0.36	0.35	0.50	0.16	0.39	0.24
10	0.03	0.27	0.21	0.00	0.18	0.04	0.01	0.28	0.13	0.02
11	0.00	0.92	0.38	0.08	0.39	0.00	0.00	0.91	0.97	0.00
12	0.02	0.67	0.99	0.00	0.70	0.20	0.13	0.19	0.25	0.61
13	0.00	0.56	0.78	0.76	0.53	0.00	0.00	0.55	0.32	0.00
14	0.33	0.00	0.01	0.00	0.01	0.38	0.09	0.04	0.00	0.30
15	0.73	0.45	0.26	0.02	0.25	0.69	0.63	0.02	0.02	0.55

FIG. 7 – Tableau d'entropie des capteurs des dix individus. Il est difficile à interpréter.

L'analyse de ces données est rendue difficile par l'absence de schéma simple. Il faut comparer les individus deux à deux pour voir les similarités entre les valeurs de leurs capteurs. C'est très fastidieux, et surtout, impossible à transformer simplement en une tâche informatique rapide.

Il est intéressant de noter que les valeurs de ces entropies restent les mêmes à 10^{-2} près lorsqu'on n'utilise que 0.5% des données (on ne relève la valeur des capteurs que tous les 200 pas de temps). Autrement dit, on peut accélérer le calcul de ces entropies par un facteur 200 sans influence sur le tableau obtenu.

Pour faciliter la comparaison entre individus il faut disposer d'une distance qui mesure leur différence : la mesure de diversité. On peut alors additionner les différences en valeur absolue des entropies de ces capteurs, mais cette somme rend tous les capteurs équivalents... Afin de se prémunir contre cette perte d'information, on choisit d'utiliser une autre mesure basée sur l'entropie : l'entropie mutuelle de deux individus. Elle se calcule de la manière suivante pour des individus 1 et 2 :

$$S_{mutuelle} = \sum_{\alpha=1}^{nombre\ de\ capteurs} \sum_{i=1..10} (p_i^{1,\alpha} - p_i^{2,\alpha})\ ln(\frac{p_i^{1,\alpha}}{p_i^{2,\alpha}})$$

De façon très similaire au cas précédent, $p_i^{1,\alpha}$ est la probabilité que le α-ième capteur de l'individu numéro 1 prenne une valeur dans l'intervalle $[0.1 * i, 0.1 * (i + 1)[$.

Cette fois le résultat est un simple nombre, qui *donne directement une mesure de diversité.*

Le tableau d'entropie mutuelle obtenu est le suivant :

Individus	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	16	17	8	16	1	4	18	29	2
2	X	2	11	2	15	15	3	6	13
3	.	X	9	1	15	16	8	6	13
4	.	.	X	13	6	7	9	15	11
5	.	.	.	X	15	16	8	9	14
6	.	.	.	.	X	3	17	24	2
7	.	.	.	.	.	X	18	26	5
8	.	.	.	.	.	.	X	17	18
9	.	.	.	.	.	.	.	X	20

FIG. 8 – Tableau à double entrée des entropies mutuelles.

Les valeurs d'entropie mutuelle inférieures à 5 correspondent à des individus à priori très proches. A partir de ces données il est extrêmement facile de voir quels individus sont proches, et lesquels sont différents. Grâce à cette méthode, j'ai pu retrouver sans aucune erreur et très rapidement la bonne réponse. En particulier, on remarque les groupes 2-3-5-8–9 et 1-6-7-10–4. Il s'avère que les individus du premier groupe étaient des robots se déplaçant en avant et ceux du second en arrière ! Ce résultat est extrêmement encourageant puisqu'il montre que l'entropie mutuelle est capable de caractériser correctement des différences de comportement à partir de données temporelles brutes extraites des capteurs.

Malheureusement, l'entropie mutuelle se calcule par paire d'individus, le nombre de calculs de diversité grandit donc en n^2 avec cette méthode... On peut penser que la relation "ressemble à" est transitive, et diminuer ainsi le nombre de calculs à réaliser, mais ce n'est pas vrai dans tous les cas... La méthode est donc très puissante, mais aussi plus coûteuse.

Finalement, on ne comparera les méthodes déjà existantes qu'avec la première méthode entropique présentée ici. Il sera important de réaliser cette comparaison avec l'entropie mutuelle.

2.3 Des performances satisfaisantes

Les différents critères de diversité utilisés dans l'expérience du ramassage de balles sont les suivants :

- Le critère Ad hoc : Ce critère très simple est spécifique au problème étudié. Il repose sur la capacité du robot à déplacer des balles. Ce critère facilite la tâche aux robots car il introduit une pression de sélection basée sur la réussite d'une partie de la tâche : savoir bouger les balles. Sa valeur est :

$$\sum_{balles} position\_balle_{t=fin} - position\_balle_{t=0}$$

- Le critère Trajectoire : Ce critère évalue la trajectoire de chaque robot et mesure une distance euclidienne entre ces trajectoires. Il est spécifique aux problèmes de robotique mobile et facilite lui aussi la tâche aux robots en favorisant une exploration importante de l'arène.

- Le critère Hamming : Ce critère est général, il s'appuie sur les mêmes données que le critère d'entropie décrit plus haut : les valeurs numériques des capteurs et effecteurs du robot. Il compare ces chaînes de caractères bit à bit entre deux individus. Ces chaînes étant très longues, cette distance est assez coûteuse en calculs. Cette distance ne facilite pas la tâche au robot en introduisant aucun biais.

- Le critère Entropie : C'est le critère décrit plus haut, il est général. Il est difficile de savoir s'il facilite la tâche aux robots ou non. Par exemple, il poussera le capteur du panier à changer de valeur souvent, donc aidera les robots à trouver le panier. Mais il a ce même effet sur tous les capteurs, et pousser, par exemple, le détecteur de l'interrupteur à changer de valeur souvent n'est pas avantageux ! En effet, une fois l'interrupteur activé, il ne peut plus être détecté, et on condamne donc son détecteur à garder la valeur 0 jusqu'à la fin de la simulation. Pour maximiser l'entropie de ce capteur, le robot préfère ne jamais activer l'interrupteur...

En lançant différents runs basés sur ces différents critères, on obtient des individus plus ou moins performants. On peut porter les fitness finales de ces individus sur un diagramme de type boîte à moustaches, le graphe obtenu est le suivant :

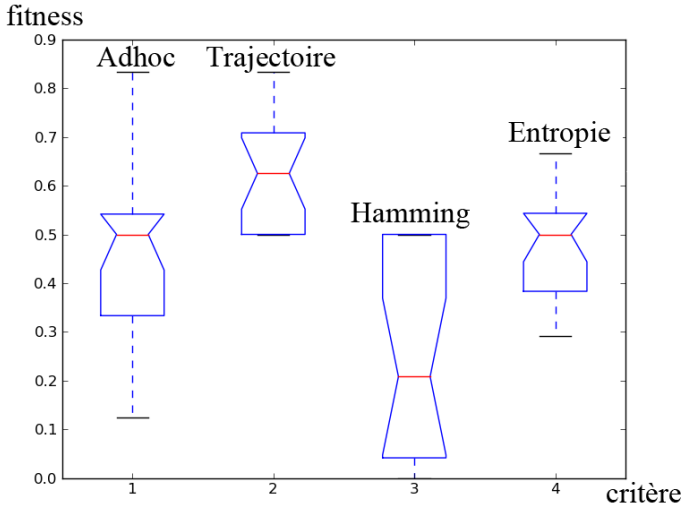

FIG. 9 – Valeurs de fitness obtenues en fonction du critère de diversité utilisé.

L'axe vertical est la valeur de la fitness, c'est en fait le nombre de balles ramassées sur le nombre de balles présentes dans l'arène. Un robot réussissant parfaitement la tâche ramasse toutes les balles de l'arène, il a donc une fitness de 1. On peut noter qu'aucune méthode n'a jamais permis d'atteindre une telle valeur de fitness.

On remarque que Hamming est le seul critère notablement peu efficace. Le critère Entropie est moins efficace que le critère Trajectoire. Comme nous l'avons précisé plus haut, l'explication tient au fait qu'encourager les robots à avoir une trajectoire différente les pousse à plus d'exploration. Ces robots auront donc plus tendance à activer l'interrupteur et pourront ainsi améliorer leur fitness. Un critère de diversité doit prouver qu'il permet une bonne exploration de l'espace des phénotypes, il n'est pas sensé faciliter la tâche demandée. Idéalement, la valeur du critère de diversité doit être une mesure objective de l'originalité d'un comportement. Ces résultats sont donc très encourageants pour le critère Entropie, qui permet d'obtenir des robots performants, alors que c'est un critère général et qui ne leur facilite que peu ou pas la tâche.

2.4 Peut-on échantillonner l'espace des phénotypes ?

L'espace des phénotypes peut être visualisé comme un espace qui est exploré par l'algorithme évolutionniste, et où chaque point correspond à un comportement. En pratique, l'algorithme génétique explore l'espace des génotypes G, mais il existe un mapping M injectif entre l'espace des génotypes et celui des phénotypes P, aussi appelé carte.

$$G \xrightarrow{M} P$$

Ainsi, à chaque mutation du génotype du robot (de son réseau de neurones), l'individu obtenu a un comportement qui correspond à un point dans l'espace P. Bien entendu,

cet espace est immense, et sa topologie très incertaine. Mais ce qui est certain, c'est que le meilleur individu, que l'on aimerait atteindre par évolution, y figure. Dès lors, une façon de trouver cet individu consiste à échantillonner de façon exhaustive cet espace ! Pour faire cela, il faut appliquer une méthode d'échantillonnage suffisamment rapide. Les algorithmes évolutionnistes permettent l'évaluation de nombreux points de cet espace : en les y disposant de façon homogène, et en augmentant leur densité dans les zones prometteuses, on maximise les chances de trouver le meilleur individu.

La partie de l'algorithme qui est chargée de la répartition homogène est l'objectif de diversité, la partie qui se charge de l'exploitation d'une zone prometteuse est l'objectif de fitness. La difficulté principale réside dans l'optimisation de notre fenêtre d'observation. En effet, si celle-ci est trop large, on a peu de chances de passer à proximité d'un bon individu, et l'on ne trouvera aucun point de départ pour l'exploitation. Mais un pas trop étroit augmente considérablement le temps de convergence.

Toutes ces considérations théoriques mériteraient peut-être d'être appliquées au travers d'une méthode nouvelle, grâce à un algorithme dont on ferait diminuer progressivement la taille de fenêtre. Cette approche serait assez radicalement différente de la façon dont fonctionnent les algorithmes évolutionnistes, mais on peut très bien penser la coupler avec l'évolution, par exemple en faisant une étape préliminaire d'exploration uniquement. Malheureusement cette méthode présente plusieurs inconvénients. Premièrement, il est très difficile de savoir de quelle taille est l'espace des phénotypes, et s'il est réaliste ou non de vouloir l'échantillonner. Deuxièmement, si l'on y parvient, il faut construire une carte de cet espace et la consulter à chaque étape, ce qui peut être coûteux en calculs. Troisièmement, il est impossible de prédire si le génotype proposé est seulement capable d'atteindre les points intéressants de l'espace des phénotypes (problème de la surjectivité). Enfin, et c'est le problème principal, toute cette méthode repose sur un critère de diversité parfait qui permette de mesurer objectivement la distance entre deux comportements dans l'espace des phénotypes.

3 Le robot plombier, optimisation en robotique évolutionniste

3.1 Présentation du problème

Les algorithmes évolutionnistes sont une méthode d'optimisation très efficace qui, comme on l'a vu, peut être appliquée au domaine de la robotique. Afin de bien maîtriser la méthode, j'ai consacré une bonne partie de mon stage à appliquer cette méthode à un nouveau problème : celui du robot plombier.

On s'intéresse à la meilleure façon de concevoir un robot qui soit apte à se déplacer dans des tuyaux de plomberie. Plusieurs éléments rendent cette tâche difficile. Les tuyaux ne sont pas en ligne droite, mais comportent des virages, ils peuvent être usés, donc ac-

cidentés, ou sales, ce qui constitue autant d'obstacles à éviter ou à enjamber ; enfin, ils contiennent des raccords, donc des changements de pente occasionnels. De plus les tuyaux peuvent être de différents diamètres, et on aimerait avoir un robot qui soit à l'aise dans tous ceux-ci.

La première question qui se pose est celle de la structure du robot. Celui-ci peut être mobile grâce à des roues, ou au contraire posséder des pattes. On peut même envisager un robot hybride possédant des roues au bout de ses pattes. L'étude de ce problème par évolution génétique permettra de savoir quelle est la meilleure solution. Il suffit en effet de générer les trois types de robot et de voir lequel est le plus rapide et le plus à l'aise dans les différents environnements où il est plongé. Une fois les structures imaginées et fixées, se pose la seconde question de la dynamique du robot. Comment sera-t-il commandé, par des moteurs ou par des servos ? Doit-il posséder un réseau de neurones ou peut-on trouver directement des commandes moteur ?

La réponse à ces questions dépend du cahier des charges. Mais dans mon cas, j'ai dû découvrir intégralement l'algorithme évolutionniste ainsi que le simulateur physique ODE. Voulant obtenir quelque chose de fonctionnel rapidement, je me suis donc contenté d'un environnement composé d'un tuyau droit, d'une structure à 4 pattes et d'une dynamique particulièrement simple. Bien entendu, maintenant que ce travail préliminaire a été réalisé, il est facile de modifier des parties de cette expérience. Par exemple, pour obtenir des comportements dynamiques crédibles il serait certainement bon d'utiliser des réseaux de neurones pour commander les servos du robot. De plus, il est très aisé de changer la structure du robot. Au final, chaque étape de la mise en place de cette expérience d'évolution génétique a été très instructive pour moi, et j'ai pu achever avant la fin de mon stage sa mise en place. Bien entendu, cela ne constitue qu'une base, et il faut à présent la réutiliser pour satisfaire le cahier des charges.

Le paragraphe suivant décrit la partie la plus difficile de cette expérience : la simulation de la physique des robots.

3.2 Modélisation du problème

Le problème du robot plombier a été modélisé en utilisant le simulateur physique ODE. Comme expliqué précédemment, ce simulateur est très rapide mais, par conséquent, n'est pas d'une grande précision. En pratique, lorsque les paramètres du simulateur sont bien réglés, les solutions obtenues sont tout-à-fait physiques et le simulateur très bon. Mais cette étape de réglage est fastidieuse, car les algorithmes évolutionnistes sont particulièrement habiles à exploiter les dysfonctionnements du simulateur : par exemple, un robot qui bondit suite à un puissant coup de pied au sol, et se satellise.

La modélisation d'un problème physique dans ODE se déroule en plusieurs étapes. On décide d'abord des éléments qui feront partie de l'environnement, et qui n'entreront pas en collision entre eux. Dans notre cas, pour commencer, cet environnement est un

simple tuyau cylindrique. Le robot devant se déplacer sur le sol de ce cylindre, on ne modélisera que la moitié inférieure de ce tuyau, car le robot n'aura aucun moyen de monter sur la moitié supérieure sans se retourner sur le dos.

Ensuite, on modélise le robot lui-même. J'ai fait plusieurs robots, l'un à pattes, l'autre à roues, et l'on passe de l'un à l'autre en changeant un simple paramètre. Le robot à pattes est constitué d'un corps et de pattes articulées en leur milieu qui y sont reliées. Les liens sont tous de type servo uniaxial dont l'amplitude et le couple maximaux sont fixés. Dans le cas du robot à roues, quatre moteurs servent à attacher les roues au corps et l'on contrôle leurs vitesses de rotation.

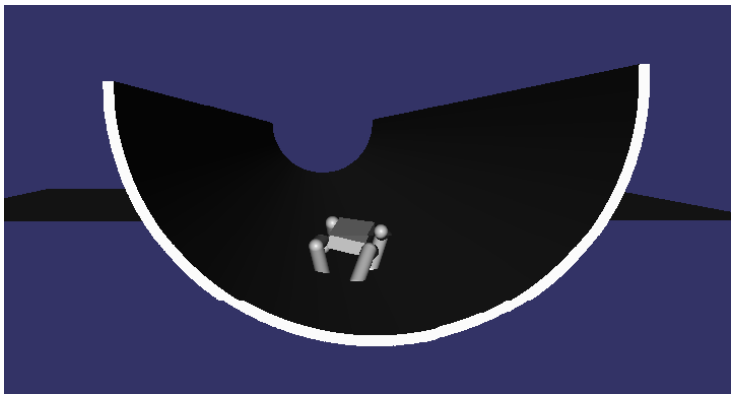

FIG. 10 – Le robot à pattes dans son environnement.

Une fois le modèle virtuel du robot créé, il faut gérer la dynamique du système. Avec ODE, des fonctions de collision existent déjà, et il suffit de les utiliser correctement. En pratique, on s'intéresse aux collisions entre le robot et son environnement uniquement. Il faut donc inclure à chaque pas de temps le calcul de ces collisions pour mettre à jour les vitesses et les positions des éléments du robot.

Une fois cette longue étape de mise en place du problème achevée, le résultat n'est rien d'autre qu'un programme en C++. Il est alors très facile de l'inclure au framework qui fera communiquer cette partie simulation avec la partie évolution.

3.3 Résultats

Dans le cadre de mon stage, je n'ai eu le temps de définir la dynamique du robot que par quelques paramètres. Je n'ai par exemple pas pu implémenter l'utilisation d'un réseau de neurones pour commander les servos du robot, ce qui aurait permis d'obtenir des comportements dynamiques plus riches et plus complexes. Pour le moment aucune solution vraiment satisfaisante n'a été trouvée, mais les modifications sont aisées et les essais de différents paradigmes dynamiques en cours. À ce jour, les recherches sur ce pro-

blème continuent.

La définition simple du comportement dynamique que j'ai utilisée consiste à donner aux pattes du robot un mouvement sinusoïdal. Toutes les parties des pattes possèdent la même fréquence, mais elles ont toutes une phase différente. De plus, les amplitudes des mouvements sinusoïdaux ne sont pas les mêmes pour la première et la seconde partie d'une patte. Ce modèle avec sinusoïdes a déjà été utilisé dans une autre expérience, et donnait des comportements assez riches malgré sa simplicité : c'est pour cela que je l'ai choisi comme point de départ.

Cette fréquence, les deux amplitudes et les huit phases sont les paramètres que l'algorithme évolutionniste doit faire évoluer. L'évolution s'effectue selon deux critères : l'optimisation est donc multi-objectifs. Ces critères sont la distance parcourue par le robot et l'énergie qu'il utilise pour faire bouger ses membres. La distance doit être maximisée, tandis que l'énergie doit être minimisée par l'algorithme.

J'ai évidemment commencé par obtenir des robots monstrueux, ou qui exploitaient un dysfonctionnement du simulateur pour se projeter à de très grandes distances. Mais si l'on fait bien tous les réglages on obtient des comportements de marche qui sont physiquement crédibles, bien que très imparfaits : deux pas en avant, un pas un arrière par exemple. Plusieurs éléments peuvent être à l'origine de ce problème. Premièrement, le modèle dynamique utilisé n'est peut-être pas assez riche pour le problème étudié. Deuxièmement, il est possible que l'évolution ne se déroule pas sur un temps assez long. Soit parce que la population n'est pas assez grande, soit parce qu'on n'effectue pas suffisamment de générations. Au final, si on veut garder un temps d'expérience raisonnable (4h actuellement), on peut faire un compromis entre ces différents facteurs. Voici une séquence d'images qui montre un comportement obtenu :

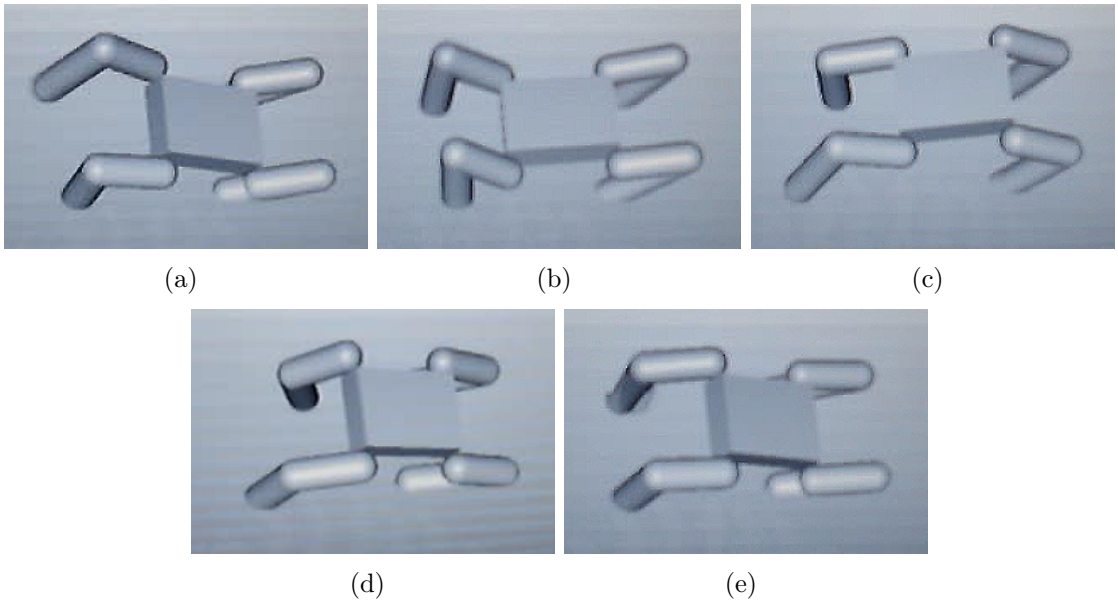

FIG. 11 – Exemple de déplacement vers la gauche par sauts successifs.

Conclusion

Ce stage d'une durée de seulement deux mois a été riche et intense. J'ai en effet découvert énormément de choses sur le domaine de la robotique et sur les méthodes évolutionnistes. De plus j'ai appris un langage de programmation qui était totalement nouveau pour moi en un temps très réduit grâce à la possibilité d'appliquer immédiatement ce que je faisais, et surtout grâce à l'accès à des programmes complexes et bien conçus. Au cours de cette période d'apprentissage j'ai pu faire les tests nécessaires à la mise en place d'un critère de diversité basé sur l'entropie. Ceci a permis par la suite d'obtenir des résultats très motivants quant à la création d'un critère de diversité efficace et général.
Mon travail sur le problème du robot plombier a lui aussi été très enrichissant, car il m'a permis de maîtriser l'utilisation d'un algrotithme évolutionniste. C'était un de mes objectifs principaux pour ce stage car j'aimerais à l'avenir appliquer ces méthodes à des problèmes issus de la physique. Je pense que de nombreux domaines de la physique pourraient tirer un grand bénéfice de ces méthodes, et je pense en particulier à une problématique issue de mon stage de M1 : l'optimisation de structures 3D.
D'autre part, ce travail que j'ai réalisé sur le robot plombier constitue les bases d'un projet de recherche qui continue à l'ISIR.

Remerciements

Je tiens tout d'abord à remercier Stéphane Doncieux, qui a rendu ce stage possible. Je le remercie aussi pour la qualité de l'encadrement qu'il m'a fourni, le trouvant toujours présent pour m'aiguiller ou pour discuter. Je remercie aussi Jean Baptiste Mouret pour sa patience et sa disponibilité lorsque j'étais bloqué par des morceaux difficiles de ses programmes.

Je remercie tout particulièrement Charles Ollion et Sylvain Koos, thésards à l'ISIR, qui m'ont très régulièrement aidé à débugger mes programmes ou m'ont aiguillé vers de nouvelles solutions. Je remercie par ailleurs tous les thésards du labo qui ont rendu ce stage aussi sympathique et enrichissant.

Références

[1] J.-B. Mouret and S. Doncieux, *Sferesv2 : Evolvin' in the multi-core world* Evolutionary Computation (CEC), July 2010, p. 1–8.

[2] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, *A fast and elitist multiobjective genetic algorithm : NSGA-II*, Evolutionary Computation, IEEE Transactions, April 2002, Vol. 6 :2, p. 182–197.

[3] S. Koos, J-B. Mouret, and S. Doncieux, *Crossing the reality gap in evolutionary robotics by promoting transferable controllers*, GECCO '10 Proceedings of the 12th annual conference on Genetic and evolutionary computation, 2010, p. 119–126.

[4] J. Lehman and K.O. Stanley, *Exploiting Open-Endedness to Solve Problems Through the Search for Novelty*, Artificial Life, 2008, ALIFE-XI p. 329–336.

[5] F. J. Gomez, *Sustaining diversity using behavioral information distance*, GECCO '09 Proceedings of the 11th Annual conference on Genetic and evolutionary computation, p. 113–120.

[6] S-H. Liu, M. Mernik and B. R. Bryant, *Entropy-driven exploration and exploitation in evolutionary algorithms*, BIOMA 2006.

[7] J. P. Rosca, *Entropy-driven adaptive representation*, Proceedings of the Workshop on Genetic Programming : From Theory to Real-World Applications, p. 23–32.