L'approche par transférabilité : une réponse aux problèmes de passage à la réalité, de généralisation et d'adaptation

Sylvain Koos

Thèse de Doctorat
de l'université Pierre et Marie Curie

Spécialité : **Informatique (EDITE)**

Présentée par : **M. Sylvain Koos**

Pour obtenir le grade de
Docteur de l'Université Pierre et Marie Curie

L'approche par transférabilité : une réponse aux problèmes de passage à la réalité, de généralisation et d'adaptation

Thèse dirigée par **Stéphane Doncieux** et **Jean-Baptiste Mouret**
soutenue le 30 novembre 2011
devant le jury composé de :

Dr. Josh Bongard	University of Vermont	Examinateur
Dr. Raja Chatila	ISIR, Univ. Pierre et Marie Curie	Examinateur
Dr. Stéphane Doncieux	ISIR, Univ. Pierre et Marie Curie	Directeur de Thèse
Dr. Jean-Baptiste Mouret	ISIR, Univ. Pierre et Marie Curie	Co-directeur de Thèse
Dr. Pierre-Yves Oudeyer	INRIA Bordeaux Sud-Ouest	Rapporteur
Dr. Marc Schoenauer	INRIA Saclay Île-de-France	Rapporteur

Résumé

Il est difficile de concevoir des contrôleurs pour des robots devant fonctionner dans des environnements peu maîtrisés voire inconnus. Dans cette optique, la robotique évolutionniste cherche à élaborer des méthodes de conception automatique de contrôleurs via un processus d'optimisation "boîte noire" utilisant des algorithmes évolutionnistes. Les valeurs de performance d'un contrôleur donné sont alors estimées soit directement sur le robot, soit à l'aide d'une simulation, par essence simplificatrice. Partant du constat qu'évaluer sur le robot et dans toutes ses situations d'utilisation est généralement incompatible avec le nombre d'évaluations requis par de tels processus d'optimisation, nous proposons une approche générale combinant un processus d'optimisation multi-objectif dans un simulateur fixe, un modèle de substitution et quelques tests sur le robot. Cette approche par transférabilité peut s'appliquer à tout processus d'optimisation mené dans un environnement simplifié (simulation) pour un environnement complet ciblé (robot) et recherche les contrôleurs qui maximisent deux objectifs : la performance dans l'environnement simplifié et un objectif de transférabilité qui indique à quel point le comportement dans l'environnement simplifié est proche de celui dans l'environnement complet. Ce deuxième objectif est estimé par un modèle de substitution construit en effectuant quelques expériences de transfert sur le robot pendant l'optimisation. L'approche est validée sur trois problèmes de robotique évolutionniste : le passage de la simulation à la réalité, l'optimisation de contrôleurs dotés de capacités de généralisation et l'adaptation d'un robot à son environnement.

Abstract

The transferability approach : an answer to the problems of reality gap, generalisation and adaptation

The design of controllers for robots that have to deal with unknown or poorly controlled environments is a difficult engineering problem. Evolutionary robotics tackles this challenge by developing automatic methods to design controllers that are based on black-box optimization processes using evolutionary algorithms. In this context, the performance values are estimated for each controller either directly on the robot or with a simulation. Assessing performance with the real robot and in all the possible situations is usually not consistent with the evaluation budgets required by these optimization methods. We here propose a general approach, which combines a multi-objective optimization process in a fixed simulation model, a surrogate model and a few experiments on the robot. This transferability approach can be applied to any optimization process conducted in a simplified environment (simulation) for a targeted full environment (robot). It looks for controllers that maximize two objectives : the performance in simulation and a transferability objective, which reflects how well the behavior observed in simulation matches the behavior on the robot. The second objective is estimated by a surrogate model built by performing a few transfer experiments on the robot while optimizing controllers. The approach is applied on three open problems from evolutionary robotics : the reality gap problem, the optimization of controllers with generalization abilities and the adaptation of a robot to its environment.

Laboratoire d'accueil

ISIR - Institut des Systèmes Intelligents et de Robotique
Université Pierre et Marie Curie
4 place Jussieu
75005 Paris - France
Encadrants : Stéphane Doncieux & Jean-Baptiste Mouret

Remerciements

Commencer un manuscrit de thèse au ton froid et impersonnel par une page de chaleureux remerciements est une contradiction. Le lecteur scientifique n'a que faire de "mercis" à des personnes qu'il ne connaît pour la plupart pas. Pourtant cette page est là, fidèle au poste, parfois brève et efficace, parfois allongée à l'eau de rose, et toujours intéressante à lire (peut-être parce qu'il s'agit du chapitre de thèse le plus court) : car la thèse est un contrat entre plusieurs êtres plein d'idées et la page de remerciements en est la signature finale, un instantané fragmentaire d'un peu plus de 3 ans de vie émaillé de quelques anecdotes bien choisies.

Je ne peux tout d'abord pas m'empêcher de remercier Stéphane et Jean-Baptiste, ne serait-ce que pour leur présence et leur disponibilité au cours de ces quelques années. Tous les doctorants n'ont pas la chance de pouvoir monter un escalier et discuter avec ses encadrants pour un oui ou pour un non (et encore, dans des temps plus anciens il me suffisait juste de tourner la tête et d'élever un peu la voix). Donc au moins pour ça, merci, et merci aussi pour tout le reste et la science entre autre.

Une thèse c'est aussi un bureau (tout du moins pour les plus chanceux). Et quel bureau puisqu'il ne compte pas moins de 9 places et a vu défiler bien plus de 9 personnes au gré des ans ! Les gens d'un même bureau partagent bon nombre de choses. En l'occurrence et par désordre d'importance : une machine à café, des idées, du sucre, un tableau blanc, des étudiants, des collègues, la douce chaleur de l'hiver au coin des machines, la cuisson estivale qui fait fondre notre ardeur à la tâche. Merci donc à Paul, Charles, Jean, Tony, Charles, Ilaria, Henry, Cédric, Lise, Nicolas et les autres.

Quand je n'étais pas dans ce bureau et pas non plus en cours ou en formation, j'étais vraisemblablement dans la salle Mouv' à regarder un e-puck désespéré cherchant à sortir d'un labyrinthe par la force ou un robot quadrupède testant la solidité du sol (qui lui a heureusement survécu). La salle Mouv' a été en quelque sorte mon lieu de vacances lors de l'été 2010 et j'ai une pensée pour tous les stagiaires qui s'y trouvaient à l'époque, sans qui cette période aurait été assez morose. Merci également à Viviane et Ludovic qui m'ont permis d'accéder à cette salle et à son équipement en toute confiance et sans contrainte. Un autre merci à Paul en passant pour m'avoir donné l'idée savante d'enlever les caches des caméras lors de mes expériences, un acte qui a sans doute eu une influence significative sur les résultats obtenus. Bien sûr, merci à Jérémie et Nicolas pour leur aide précieuse et leur enthousiasme.

Un grand merci à mes étudiants qui, à défaut d'avoir eu le choix de faire ma connaissance, semblent le vivre assez bien. Merci également à mes collègues d'enseignement, en particulier Xavier, Christophe et Marie-Jeanne.

Enfin, je voudrais remercier Ugo, Antoine et Guillaume qui, pour des raisons tout à fait différentes, m'ont fourni l'occasion d'écrire ces quelques pages.

Table des matières

Chapitre 1

Introduction

> There is a real danger (in fact, a near certainty) that programs which work well on simulated robots will completely fail on real robots because of the differences in real world sensing and actuation.
>
> Rodney A. Brooks — *Artificial Life and Real Robots* — 1992

Concevoir des contrôleurs pour des robots confrontés à des environnements inconnus ou mal maîtrisés est un problème d'ingénierie difficile [Brooks 1991a, Brooks 1992, Harvey *et al.* 1993]. Si l'on connaît un ensemble d'équations différentielles décrivant la dynamique du robot dans l'environnement considéré, il est possible d'en dériver des contrôleurs optimaux ou robustes via des méthodes de la théorie du contrôle (voir par exemple [Canudas-de-Wit *et al.* 1996, Sciavicco & Siciliano 2000, Bubnicki 2005]). Quand une telle description précise du robot n'est pas disponible, par exemple quand les caractéristiques de l'environnement ne sont pas complètement connues, les contrôleurs obtenus via de telles méthodes risquent de ne pas toujours être efficaces sur le robot du fait de simplifications dans le modèle de dynamique (voir par exemple [Adigbli *et al.* 2007, Lucet *et al.* 2009]).

En parallèle de ces approches dites classiques, des chercheurs de plusieurs domaines de recherche dont la robotique évolutionniste se sont intéressés depuis la fin du $20^{ème}$ siècle à la possibilité de définir des méthodes de conception automatiques de contrôleurs pour des robots effectuant des tâches utiles dans des environnements non structurés [Brooks 1991b, Cliff *et al.* 1993b]. Dans ce cadre, la performance d'un contrôleur est évaluée par une expérience, sur le robot ou via un modèle de simulation et le contrôleur peut potentiellement exploiter toutes les spécificités sensorielles et motrices du robot pour résoudre le problème considéré de manière originale. La fonction de performance ou *fitness* indique les caractéristiques quantitatives ou qualitatives que le comportement du robot doit posséder pour être efficace et un algorithme évolutionniste est utilisé pour trouver les contrôleurs qui maximisent cette performance (voir par exemple [Nolfi & Floreano 2000]). Une fois la fonction de performance définie, le processus se déroule de manière totalement automatique et le concepteur n'a ni à indiquer les équations différentielles de la dynamique du robot, ni à définir ce que le robot doit faire dans telle ou telle situation sensorielle.

Une question-clé dans ce type de processus est la localisation de la phase d'évaluation. Deux approches principales ont été proposées : (1) soit le contrôleur est évalué sur le robot [Brooks 1991a, Brooks 1991b, Floreano & Mondada 1994, Yosinski *et al.* 2011], (2) soit il est évalué dans un simulateur du robot et de son environnement [Jakobi *et al.* 1995, Jakobi 1998a, Floreano & Urzelai 2001, Saunders *et al.* 2011]. Rodney Brooks fut l'un des premiers à proposer des méthodes automatiques de conception de contrôleurs de robots, en particulier dans le cadre de travaux d'intelligence artificielle [Brooks 1991b]. Une des hypothèses qu'il soutient dans plusieurs articles est que l'évaluation de la performance d'un contrôleur doit se faire de manière incarnée sur le robot [Brooks 1991a, Brooks 1991b, Brooks 1992] car l'utilisation d'une simulation ne serait pas suffisamment fiable.

L'hypothèse de Brooks repose en partie sur le constat qu'il peut être difficile de modéliser correctement ce qu'un robot perçoit de son environnement et quelles informations il peut effectivement exploiter : les valeurs produites par ses capteurs, parfois entachées de fortes incertitudes, ne fournissent pas une description directe des objets contenus dans le monde [Brooks 1992]. En évaluant le contrôleur sur le robot, il est confronté à la fois à la perception réelle du robot et à la dynamique concrète de l'environnement et peut potentiellement exploiter des phénomènes très spécifiques sur sa morphologie ou sa dynamique qu'un modélisateur omettrait d'intégrer à un simulateur. Cette méthodologie est d'autant plus attractive que de nombreux travaux utilisant une simulation pour évaluer des contrôleurs de robots sont obligés d'apporter une réponse à ces problèmes de passage à la réalité ou *reality gap* (pour différents exemples, voir [Nolfi *et al.* 1994, Jakobi *et al.* 1995, Miglino *et al.* 1995, Jakobi 1998a, Funes & Pollack 1999, Urzelai & Floreano 2001, Floreano & Urzelai 2001]). D'un point de vue plus général, ces problèmes de passage de à la réalité ne sont pas spécifiques à la conception automatique de contrôleurs robotiques. Ils peuvent intervenir dans tout travail d'optimisation ou d'apprentissage où la performance des solutions est obtenue dans un environnement simplifié (la simulation) par rapport à l'environnement final (le système ciblé). Dans ce cadre plus général, l'hypothèse de Brooks pourrait être étendue de la manière suivante : *l'évaluation des solutions doit s'effectuer sur le système ciblé dans son environnement complet et non dans un environnement simplifié.*

Avant de pouvoir s'interroger sur l'intérêt d'évaluer ou non sur le robot, il faut fixer le cadre des expériences robotiques que nous allons considérer. Tout d'abord, aucun a priori n'est fait sur le système étudié et la performance des contrôleurs est maximisée sans s'intéresser au détail du processus d'évaluation : ces deux contraintes nous placent dans un problème optimisation boîte noire (e.g. [Hebbel *et al.* 2006, Hemker *et al.* 2009]). En conséquence, la qualité d'un contrôleur ne peut être estimée qu'en évaluant celui-ci via une expérience, que ce soit sur le robot ou dans un simulateur. Une fonction définie par le concepteur permet ensuite de calculer une ou plusieurs valeurs de performance pour chaque contrôleur à partir du comportement correspondant dans l'environnement d'évaluation. Pour trouver des solutions d'intérêt, la méthode de conception automatique va chercher les contrôleurs qui maximisent cette ou ces valeurs. Dans le cas où plusieurs fonc-

tions de performance doivent être optimisées simultanément, le problème d'optimisation est dit multi-objectif (e.g. [Pires *et al.* 2004, Molina-Cristóbal *et al.* 2005]). Si par exemple le robot doit se déplacer le plus rapidement possible tout en dépensant un minimum d'énergie, il peut alors exister différents compromis possibles entre la vitesse moyenne et l'énergie dépensée : la sélection de la solution finale se fera parmi les contrôleurs correspondant aux meilleurs compromis vitesse-énergie. Dans ce contexte, nous choisissons d'optimiser les contrôleurs via un algorithme évolutionniste étant donné qu'il s'agit d'une méta-heuristique efficace à la fois pour traiter des problèmes d'optimisation boîte noire (voir par exemple [Spall 2003]) et pour trouver un ensemble de compromis d'un problème multi-objectif (voir [Fonseca & Fleming 1995] et [Deb 2001a]). Un tel choix place mécaniquement ce travail dans le domaine de la robotique évolutionniste qui regroupe l'ensemble des travaux appliquant des algorithmes évolutionnistes à des problèmes robotiques.

D'après l'hypothèse de Brooks, pour connaître la performance d'un contrôleur donné, il faudrait théoriquement le tester sur le robot dans l'ensemble des environnements et des contextes de la tâche à résoudre qu'il va potentiellement rencontrer. Néanmoins, concevoir des contrôleurs en les évaluant directement sur le robot sans passer par un simulateur présente plusieurs inconvénients dans le contexte défini précédemment.

- Les expériences sur le robot sont contraintes au temps réel. Un processus d'optimisation effectué directement sur le robot ne pourra donc pas bénéficier de la loi de Moore [1] [Moore 1975, Schaller 1997], c'est-à-dire de l'augmentation de la puissance de calcul des ordinateurs. *A contrario*, un processus où l'évaluation des solutions s'effectue en simulation pourrait en profiter.
- Les techniques d'optimisation peuvent nécessiter de nombreuses évaluations de contrôleurs [Brooks 1992], ce qui signifie de nombreuses expériences difficilement parallélisables sur le robot avant de trouver une solution efficace. Un processus d'optimisation correct pourra difficilement être obtenu avec des contraintes drastiques sur le nombre d'expériences.
- Le comportement d'un contrôleur n'étant pas connu a priori avant son évaluation, chaque expérience menée sur le robot est potentiellement dangereuse.
- Pour s'assurer qu'un même contrôleur fonctionnera de manière robuste dans un environnement donné, il peut être nécessaire d'effectuer plusieurs évaluations dans des contextes différents de la tâche (par exemple, en changeant la position initiale du robot). Une telle contrainte peut dans certaines applications multiplier le nombre d'évaluations total à effectuer et rendre impossible une évaluation systématique des solutions sur le robot.

Optimiser des contrôleurs sur un robot dans tous les contextes possibles est un processus coûteux en temps expérimental et difficilement réductible autrement qu'en utilisant plusieurs copies du robot (e.g. [Bongard 2007]). En optimisant sur le ro-

1. La première loi de Moore telle qu'elle fut formulée en 1975 indique que le nombre de transistors par microprocesseur double tous les deux ans (voir [Moore 1975] pour la formulation exacte). Cette loi empirique est approximativement vérifiée depuis les années 1970.

bot, il n'est généralement pas envisageable d'assurer en pratique le même nombre d'évaluations que pour d'autres applications typiques des algorithmes évolutionnistes : de quelques centaines à quelques milliers d'évaluations en robotique évolutionniste (e.g. [Floreano & Mondada 1998, Zykov *et al.* 2004, Yosinski *et al.* 2011]) contre plusieurs dizaines de milliers dans le cadre d'optimisations multi-objectif (e.g. [Zitzler *et al.* 2000, Deb *et al.* 2002b]) ou plusieurs millions pour des expériences de neuro-évolution (e.g. [Ruppin 2002]). Pour pouvoir proposer un nombre d'évaluations plus adéquat dans un temps limité, il est donc nécessaire que la majeure partie du processus d'optimisation s'effectue dans un environnement simplifié, et ce malgré les problèmes de passage à la réalité. Dans un cadre plus général, il est donc nécessaire de répondre à la question suivante :

Comment obtenir des solutions performantes dans l'environnement complet en les optimisant principalement dans un environnement simplifié ?

Le but de ce manuscrit est d'apporter une réponse à cette question générale via une nouvelle méthodologie d'optimisation reposant à la fois sur l'utilisation d'un simulateur fixe permettant d'évaluer rapidement les solutions, tout en autorisant un nombre très limité d'expériences sur le robot. Dans ce contexte, l'approche proposée peut se formuler comme un problème d'optimisation multi-objectif où les solutions doivent maximiser à la fois la performance dans l'environnement simplifié et la performance dans l'environnement complet. Cette seconde fonction de performance est approximée via un modèle de substitution (ou *surrogate model*) au cours du processus d'optimisation à l'aide de tests de quelques contrôleurs pertinents sur le robot. L'optimisation est alors guidée vers des solutions effectivement efficaces dans l'environnement complet.

Cette méthodologie sera validée dans trois cas d'étude tirés de la robotique évolutionniste où la question du passage d'un environnement simplifié à l'environnement ciblé est cruciale (cf. figure 1.1) :

A. **Comment optimiser un contrôleur en simulation qui se transfère bien sur le robot physique ?** Cette question sera notamment traitée à l'aide de deux expériences, l'une de navigation avec un robot mobile e-puck et l'autre d'optimisation de comportements de marche sur un robot quadrupède. Dans les deux cas, des comportements performants sont obtenus sur le robot en autorisant moins de 10 expériences pendant l'optimisation.

B. **Comment obtenir des contrôleurs de robot qui généralisent dans de nouvelles situations ?** L'étude portera sur deux expériences sur des robots mobiles en simulation, l'une de navigation dans un labyrinthe et l'autre implémentant une tâche de collecte de balles dans une arène. L'approche permet ici d'obtenir en un temps expérimental restreint des contrôleurs robustes à des variations importantes de l'environnement du robot.

C. **Comment obtenir un robot qui s'adapte à des changements d'environnement ?** Quand le comportement d'un robot n'est plus efficace du fait d'un changement d'environnement, il est possible d'utiliser un processus

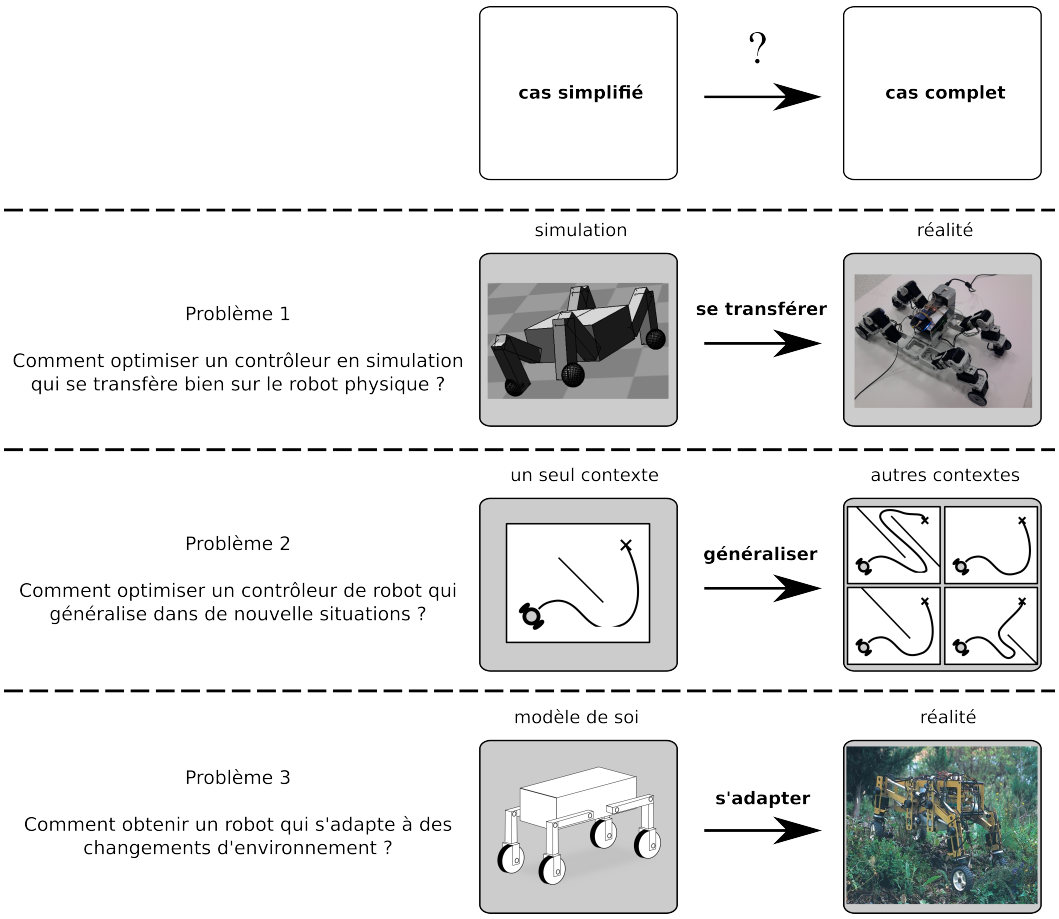

FIGURE 1.1 – Trois problèmes tirés de robotique évolutionnistes impliquant de passer d'un environnement simplifié à un environnement complet : passage à la réalité, généralisation et adaptation.

d'optimisation via un algorithme évolutionniste pour découvrir un contrôleur performant à partir d'un modèle de soi du robot. Nous verrons qu'un tel mécanisme d'adaptation peut être reformulé comme un problème de transférabilité entre le modèle de soi du robot d'une part et le robot physique dans le nouvel environnement d'autre part. Le cas particulier de l'adaptation à l'environnement d'un robot à locomotion hybride roues-pattes sera considéré. Le processus d'adaptation proposé permet au robot de découvrir en quelques minutes des comportements performants sur différents types de terrains.

Ce manuscrit est découpée en 7 chapitres. Suite à cette introduction, le chapitre II introduira les algorithmes évolutionnistes, en insistant particulièrement sur la problématique de l'optimisation multi-objectif et sur leur utilisation dans des applications robotiques. Le chapitre III présentera les modèles de substitution dans un cadre d'optimisation, un des outils importants sur lequel est basée notre approche.

Les chapitres IV, V et VI détailleront respectivement les algorithmes, résultats et réflexions menés sur chacun des trois problèmes de robotique évolutionniste indiqués plus haut : passage de la simulation à la réalité, généraliser dans de nouvelles situations et s'adapter à un changement d'environnement. Ces chapitres seront suivis d'une discussion plus générale dans le chapitre VII, axée notamment sur l'apport de notre approche à la robotique.

Chapitre 2

Algorithmes évolutionnistes multi-objectifs et robotique

2.1 Algorithmes évolutionnistes

2.1.1 Principes et inspirations

A la fin des années 1950, Alex Fraser propose d'utiliser les ordinateurs afin de simuler différents mécanismes biologiques [Fraser 1957a, Fraser 1957b, Fraser 1958, Fraser 1960]. En particulier, certains de ses travaux traitent de la simulation du processus de sélection génétique sur une population de "génomes" digitaux [Fraser 1958, Fraser 1960]. Bien que le but de Fraser soit de simuler un processus biologique, l'algorithme itératif qu'il utilise présente des points communs avec les futurs algorithmes évolutionnistes et ses différents travaux font partie des contributions séminales à l'évolution artificielle.

L'appellation algorithmes évolutionnistes [Bäck 1996, Fogel 2000, De Jong 2006] regroupe un ensemble d'algorithmes qui s'inspirent du processus d'évolution artificielle. Leur utilité s'est affirmée à partir des années 1970, notamment avec l'apparition de trois classes d'algorithmes [1] : les stratégies d'évolution [Rechenberg 1973, Schwefel 1981, Bäck *et al.* 1991], les programmes évolutionnistes [Fogel *et al.* 1966, Fogel 1995] et les algorithmes génétiques [Holland 1975, Mitchell 1998]. Sans entrer dans le détail de chaque type d'algorithme, tous présentent les points communs suivants : ils manipulent une population de solutions, une solution peut subir des modifications aléatoires, un processus de sélection permet de déterminer quelles solutions conserver et quelles solutions rejeter.

Les algorithmes évolutionnistes s'appuient sur une abstraction du principe de sélection naturelle inhérente à la théorie de l'évolution de Darwin [Darwin 1859]. Ce principe peut être résumé comme suit : les individus les plus adaptés à leur environnement dans une population ont plus de chance de se reproduire et donc de répandre leurs caractéristiques phénotypiques dans la population. Le processus de sélection naturelle est basé sur les deux aspects suivants.

- Les traits phénotypiques liés à la survie sont héréditaires, c'est-à-dire qu'un parent et son enfant devraient avoir une adaptation à leur environnement semblable.

1. Pour une description de chacune de ces classes, se référer par exemple à l'article [Bäck & Schwefel 1993].

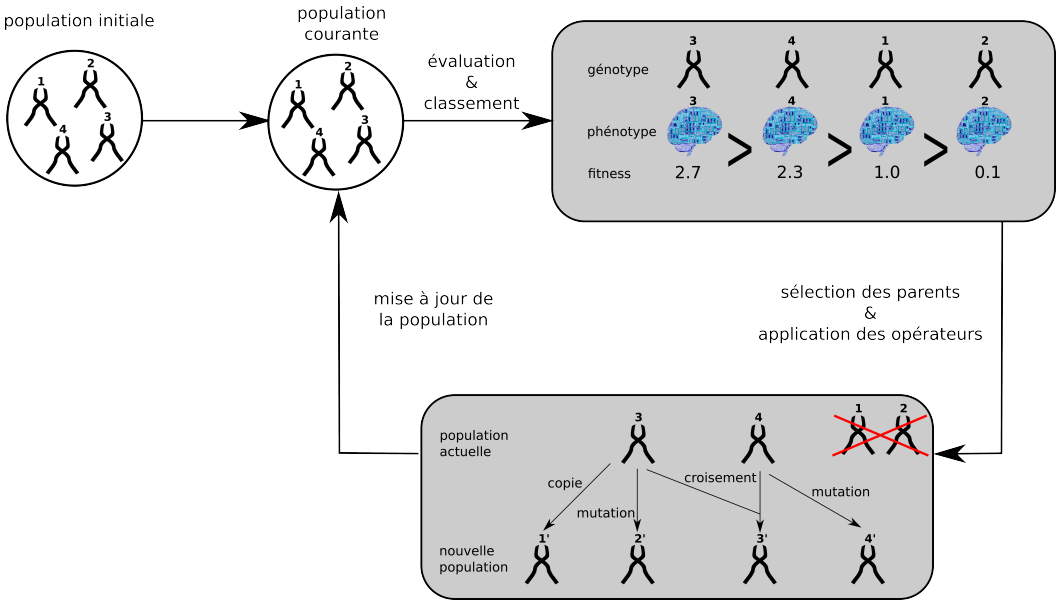

FIGURE 2.1 – Schéma d'un algorithme évolutionniste classique optimisant une population de solutions génération par génération.

- L'information génétique transmise par un parent à son enfant subit parfois des modifications, ce qui peut entraîner des variations phénotypiques, qu'elles soient délétères, neutres ou avantageuses.

Dans la suite de ce manuscrit, nous ne considérerons que des algorithmes évolutionnistes manipulant une population de solutions et où le processus d'optimisation est découpé en générations. Chaque génération de l'algorithme suit le schéma de la figure 2.1. A chaque génération, chaque individu de la population est *évalué* dans un environnement donné simulé ou réel et une note de performance ou *fitness*[2] lui est décernée en fonction du problème à résoudre (e.g. [Nelson *et al.* 2009]). Les valeurs de fitness permettent ainsi de classer les individus suivant leurs performances. Un processus de *sélection* basé sur ce classement choisit ensuite un groupe d'individus-parents (par exemple, les plus performants) utilisés pour générer la population de la génération suivante. L'étape de *reproduction*, i.e. l'obtention de la nouvelle population à partir des individus-parents, fait intervenir divers mécanismes mimant une variabilité génotypique (mutation, croisement, ...), certaines variations pouvant mener à des enfants plus performants. Ces trois étapes – évaluation, sélection et reproduction – sont itérées jusqu'à trouver un individu suffisamment performant ou après un nombre arbitraire de générations.

Les algorithmes évolutionnistes peuvent être employées en optimisation en tant que méta-heuristiques stochastiques. Comme ils n'imposent pas de contraintes

2. Le terme anglais *fitness* étant très utilisé dans la communauté des algorithmes évolutionnistes, nous l'utiliserons tel quel dans la suite de ce manuscrit.

particulières sur la fonction de performance à optimiser, ils sont notamment appliqués à des problèmes d'optimisation boîte noire [Bäck & Schwefel 1993, Wolpert & Macready 1997, Osyczka & Osyczka 2002, Spall 2003], par exemple dans le cas où les dérivées de la fonction de performance ne sont pas calculables ou encore quand la performance d'une solution ne peut être obtenue que via une expérience sur un système. Nous verrons dans la partie 2.2 de ce chapitre qu'ils permettent également de gérer efficacement les problèmes d'optimisation multi-objectifs [Fonseca & Fleming 1995, Deb 2001a, Osyczka & Osyczka 2002]. Ces deux aspects rendent ce type d'algorithmes particulièrement intéressants en robotique en général et en robotique évolutionniste en particulier, ce que nous détaillerons dans la partie 2.3.

Afin de mimer le processus de sélection naturelle, un algorithme évolutionniste est basé sur trois concepts importants que nous allons détailler tour à tour.

1. Une solution possède plusieurs niveaux de description, de sa représentation digitale ou génotype à sa valeur de performance ou fitness.
2. Des opérateurs de variabilités permettent de modifier le génotype des solutions.
3. Une ou plusieurs pressions de sélection sont appliquées à la population afin de guider l'optimisation des solutions vers des zones d'intérêt (par exemple, de hautes performances).

Dans les trois parties suivantes, nous allons donc nous concentrer sur ces différents concepts. Tout d'abord, nous détaillerons les différents niveaux de description d'un individu que sont le génotype, le phénotype et la fitness. Nous nous attarderons ensuite sur les opérateurs de variabilité génotypique habituellement utilisés, avec de définir les notions de pression de sélection et de diversité.

2.1.2 De la description d'un individu : génotype, phénotype et fitness

Dans un algorithme évolutionniste, un individu peut généralement être décrit d'au moins trois façons. A bas niveau, il possède un génotype où toute l'information relative à l'individu est encodée, que ce soit sous forme de chaînes de bits, d'un vecteur de valeurs réelles ou encore d'un graphe. Les opérateurs de variabilité génétique sont définis à ce niveau et dépendent du codage choisi.

Afin d'évaluer l'individu, son génotype doit être décodé en un phénotype. On appelle cette phase le développement du génotype. Le phénotype est la solution en tant que telle et peut directement interagir avec l'environnement d'évaluation. Dans certains cas, la phase de décodage peut être complexe, par exemple en transformant une chaîne de bits ou un graphe en réseau de neurones. Pour beaucoup d'applications, une solution peut être encodée sous la forme d'un vecteur de paramètres et le passage du génotype au phénotype revient à intégrer les valeurs des paramètres spécifiques dans la structure fixe du phénotype (paramètres d'une fonction, poids d'un réseau de neurones, ... ; e.g. [Bäck & Schwefel 1993, Deb *et al.* 2002a]). Dans certains cas, le génotype paramètre également la structure du phénotype,

ce qui donne lieu à une phase de décodage plus complexe (e.g. [Jakobi 1998a] et [Stanley & Miikkulainen 2002]). En optimisation multi-objectifs, l'espace phénotypique est aussi appelé espace de décision, étant donné que le choix final de la solution se fera dans cet espace, par exemple en fonction de contraintes sur les valeurs des paramètres [Deb 2001a].

Le phénotype est évalué dans un environnement donné et une note de fitness lui est associé. La fonction de fitness est définie par l'utilisateur et permet de calculer, pour chaque solution considérée, une valeur reflétant sa performance sur le problème (e.g. [Nelson *et al.* 2009]). La définition de la fonction de fitness est une étape importante, étant donné que les étapes de classement et de sélection d'un algorithme évolutionniste reposent souvent directement sur les valeurs de fitness observées. La fonction de fitness permet donc de définir une relation d'ordre sur les génotypes et le gradient que le processus d'optimisation doit suivre. Cette fonction peut être très détaillée en imposant différentes contraintes sur la solution optimale ou bien seulement donner une note globale au regard du problème à résoudre.

2.1.3 De la variabilité génotypique : mutation et croisement

Grossièrement inspirés de la biologie, les opérateurs de variabilité utilisés par les algorithmes évolutionnistes sont au nombre de deux : la mutation [Bäck & Schwefel 1993, Eiben & Schippers 1998] et le croisement [Eiben *et al.* 1998, Jansen & Wegener 1999].

L'apparition de mutations est un phénomène courant qui se produit lors de la réplication de la molécule d'ADN d'une cellule. Pendant le processus, certaines bases azotées qui servent de support à l'information génétique peuvent être ponctuellement modifiées, des gènes peuvent être dupliqués ou des parties de l'ADN totalement supprimées. Les algorithmes évolutionnistes utilisent principalement la mutation ponctuelle comme illustré sur la figure 2.2-A. Par exemple, en supposant que le génotype soit une chaîne de bits, on peut définir pour chaque bit une probabilité d'inversion : le génotype de l'individu muté peut alors différer de celui de l'individu original de plusieurs bits.

L'opérateur de mutation induit théoriquement de petites variations phénotypiques de par son action sur le génotype. On peut alors définir le voisinage d'un individu comme l'ensemble de ses mutants potentiels. En pratique, l'effet de la mutation est très variable et une faible variation génotypique peut entraîner des phénotypes très différents du fait de la phase de développement. En fonction du codage utilisé, il est possible de définir d'autres types de mutation – duplication d'une partie du génotype, addition, délétion, mutation étendue d'une partie du génotype, ...–, la plupart du temps inspirés de phénomènes biologiques. Nous ne détaillerons pas ces opérateurs.

Le second opérateur classique est appelé croisement, nom du phénomène biologique dont il est inspiré. Le croisement ou enjambement correspond à un phénomène où deux chromosomes s'entrecroisent et échangent une partie de leur brin d'ADN comme illustré sur la figure 2.2-B. *In silico*, l'opérateur de croisement dépend es-

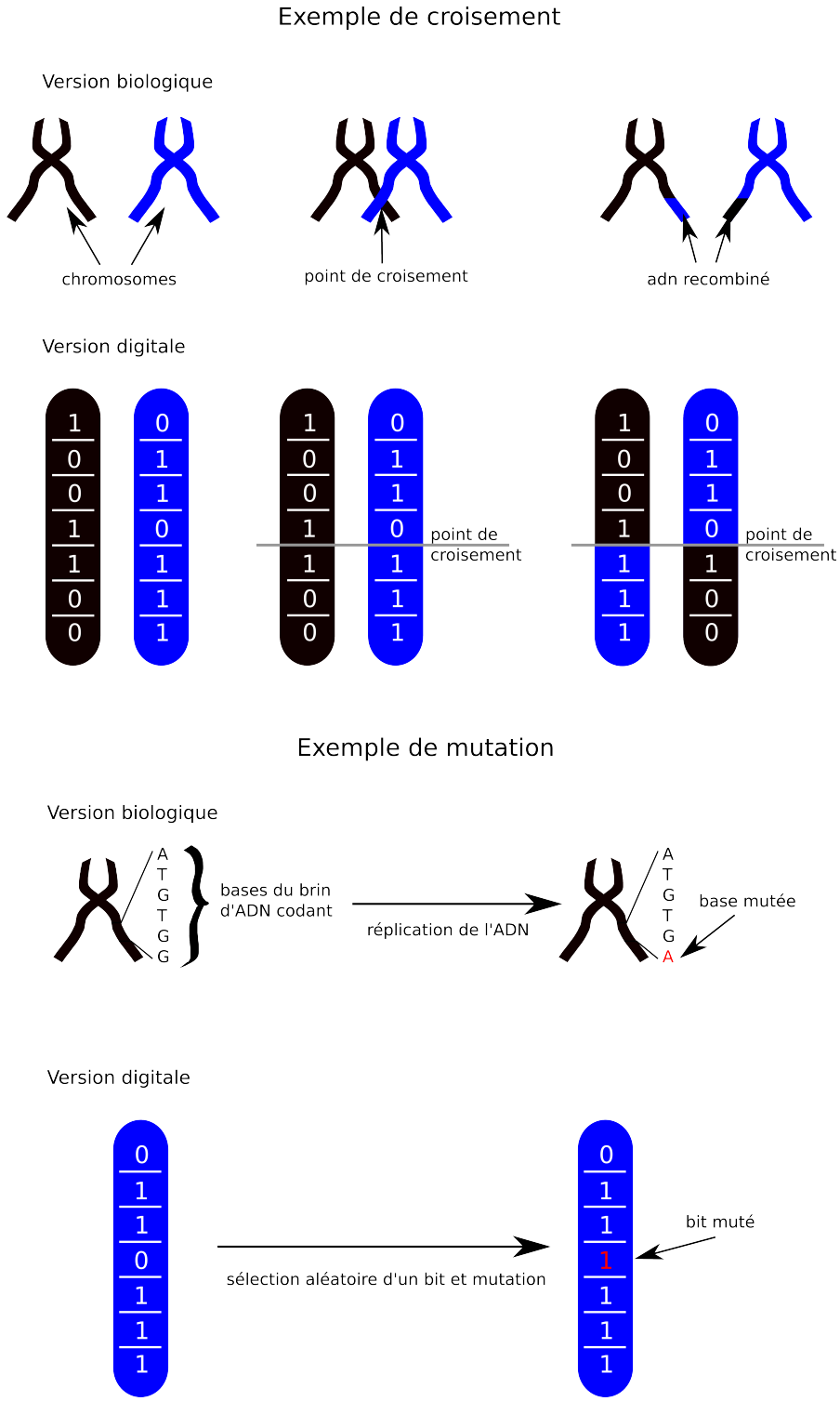

FIGURE 2.2 – Illustration des opérateurs de mutation et de croisement utilisés dans un algorithme évolutionniste, ainsi que des phénomènes biologiques dont ils sont inspirés.

sentiellement du codage utilisé. Par exemple, si le génotype est un vecteur de n éléments, le croisement entre des individus a et b revient généralement à choisir aléatoirement un nombre x entre 1 et n et créer deux nouveaux individus a' et b' tel que a' (respectivement b') contienne les x premiers éléments de a (respectivement b) et les n-x derniers éléments de b (resp. a). Il est également possible de définir un croisement multi-point.

Là encore, contrairement à la biologie où le croisement permet un brassage génétique des allèles présents sur les deux brins d'un même chromosome, l'opérateur de croisement *in silico* entraîne souvent un effet drastique sur le phénotype de la solution suivant le codage utilisé. Certains travaux semblent indiquer que l'utilisation d'un opérateur de croisement rend l'algorithme plus efficace vis-à-vis de l'optimisation [Jansen & Wegener 1999]. Il peut néanmoins être difficile de définir un opérateur de croisement qui ait un sens pour le codage utilisé, par exemple dans le cas d'évolution de graphes [Eiben & Schippers 1998, Stanley & Miikkulainen 2002]. Certains travaux se limitent même à ne définir que des opérateurs de mutation (e.g. [Bäck *et al.* 1991, Mouret & Doncieux 2011]).

2.1.4 Sélection et Diversité

Un algorithme évolutionniste introduit généralement deux étapes de sélection : (1) la sélection dans la population d'individus-parents utilisés pour générer un certain nombre d'individus-enfants ; (2) la sélection des individus de la nouvelle population parmi les individus-enfants générés ou parmi l'ensemble des individus-parents et des individus-enfants. Pour la première étape de sélection, on peut distinguer différents schémas principaux [Back 1994, Blickle & Thiele 1996, Whitley 2001] :

- une sélection par tournoi [Brindle 1981, Miller & Goldberg 1995] ;
- une sélection proportionnelle au rang ou à la fitness [Hancock 1994, Baker 1985].

La sélection par tournoi [Brindle 1981, Miller & Goldberg 1995] consiste à effectuer un nombre arbitraire de tournois entre n individus ($n \geq 2$) et de choisir le ou les plus performants. Plus n augmente, plus ce type de sélection induit une pression importante, étant donné que le tournoi a plus de chance d'impliquer les individus de la population ayant les plus hautes valeurs de fitness.

La sélection proportionnelle [Hancock 1994, Baker 1985] utilise une fonction de probabilité définie par l'utilisateur et qui dépend de la valeur de fitness. La pression imposée par une telle méthode dépend directement de la fonction choisie et varie d'une génération à l'autre : typiquement, plus les individus de la population ont des fitness différentes, plus la pression de sélection sera forte. Une variante revient à baser la fonction de probabilité sur le rang des solutions et non leur performance, ce qui rend la sélection moins dépendante des valeurs de fitness. Dans les deux cas, la fonction de probabilité est souvent linéaire en fonction de la fitness ou du rang.

La seconde étape de sélection de l'algorithme va choisir les individus à garder dans la population utilisée à la génération suivante. Cette seconde étape de sélection peut être assurée, comme précédemment, par une sélection par tour-

noi ou proportionnelle. Une autre possibilité est de garantir une sélection élitiste [De Jong 1975, Rudolph 1996], i.e. où les meilleurs individus générés sont systématiquement conservés. Dans ce cas, les individus les moins performants ne pourront pas se maintenir dans la population. Le choix du mécanisme assurant cette seconde sélection peut avoir une grande influence sur les caractéristiques de convergence de l'algorithme [Goldberg & Deb 1991, Thierens & Goldberg 1994]. Plus la sélection avantage les individus performants (i.e. de valeurs de fitness élevées) au détriment des autres, plus la pression de sélection induit par l'objectif à optimiser sera forte.

Intuitivement, la force de la pression de sélection dans cette seconde étape est reliée à la convergence du processus d'optimisation [Back 1994, Hancock 1994, Eiben & Schippers 1998, Whitley 2001] : une pression de sélection trop faible ne sélectionnera pas systématiquement les individus de meilleurs fitness, ce qui peut ralentir la convergence de l'algorithme vers les solutions optimales. *A contrario*, une pression trop forte peut rapidement entraîner l'algorithme vers des optima locaux. Ces problèmes nécessitent alors une pression de sélection qui fasse compromis entre exploration et exploitation [Bäck & Hoffmeister 1991, Mauldin 1984, Blickle & Thiele 1996, Ursem 2002, Mouret 2008] : en exploitant uniquement les individus performants, la diversité des solutions dans la population en pâtit, limitant la capacité de l'algorithme d'échapper aux effets locaux du gradient. Ce compromis exploitation-exploration n'est pas une spécificité des algorithmes évolutionnistes et intervient dans tout processus d'optimisation [Chen *et al.* 2009].

Comme il est difficile de régler la pression de sélection de l'algorithme par rapport au problème à résoudre, une pratique courante est de maintenir explicitement la diversité dans la population [Mauldin 1984, Mahfoud 1997, Mouret & Doncieux 2011]. Pour ce faire, il existe de nombreuses méthodes [Mahfoud 1997] et nous n'en détaillerons que 3 principales ici :

- le *fitness sharing* ou partage de fitness [Holland 1992, Goldberg & Richardson 1987] ;
- les méthodes de *crowding* [De Jong 1975, Mahfoud 1997] ;
- l'utilisation d'un objectif de diversité [De Jong *et al.* 2001, Toffolo & Benini 2003, Mouret & Doncieux 2011].

La première technique, appelée partage de fitness ou *fitness sharing* [Holland 1992, Goldberg & Richardson 1987], consiste à diminuer la fitness d'un individu s'il existe beaucoup d'autres individus proche de celui-ci dans la population. Par exemple, en supposant qu'une fonction de distance d entre individus (dont la définition sera discutée dans la partie suivante) existe, on peut calculer une valeur de *sharing* $sh(p,\ q)$ pour tout couple d'individus $(p,\ q)$ de la population $\mathbb{P}$:

$$sh(p,\ q) = \begin{cases} 1 - \left(\dfrac{d(p,\ q)}{\sigma_{share}}\right)^{\alpha}, & \text{si } d(p,\ q) < \sigma_{share} \\ 0, & \text{sinon} \end{cases}$$

Soit f la fonction de fitness correspondant au problème à résoudre, on peut alors définir une nouvelle fonction de fitness f_{sh} qui prend en compte la valeur de *sharing* :

$$f_{sh}(p) = \frac{f(p)}{\sum_{q \in \mathbb{P}} sh(p,\ q)}$$

La constante σ_{share} est le rayon de *sharing* et définit un voisinage (parfois appelé niche) autour de l'individu p : si des individus se trouvent dans ce voisinage, $f_{sh}(p) < f(p)$, sinon $f_{sh}(p) = f(p)$. Du point de vue de la diversité, tous les individus d'une même niche sont considérés comme équivalents.

Cette méthode est évidemment très dépendante de la valeur du rayon de *sharing* σ_{share}. Plus σ_{share} est petit, moins la population sera poussée à rester diverse : on exploite trop. Par contre, si σ_{share} est trop grand, l'optimisation risque de négliger des zones d'intérêt, les solutions étant poussées à être très éloignées les unes des autres : on explore trop.

La technique de *crowding* [De Jong 1975, Mahfoud 1997], quant à elle, ne nécessite pas de rayon de *sharing* et influe directement sur l'étape de sélection des solutions. L'idée est qu'un nouvel individu qu'on souhaite ajouter à la population va remplacer un individu qui lui est similaire. Pour ce faire, on compare le nouvel individu à n individus tirés aléatoirement dans la population et celui dont la distance au nouvel individu est la plus faible est remplacé. Là encore la notion de similarité est dépendante d'une fonction de distance d et nous verrons plus tard comment la définir.

Une alternative est de considérer la diversité comme un objectif au même titre que la fitness. On cherche alors à optimiser simultanément deux objectifs, ce qui peut être directement fait par l'utilisation d'algorithmes évolutionnistes multi-objectifs [De Jong *et al.* 2001, Toffolo & Benini 2003, Mouret & Doncieux 2011]. Le fonctionnement de tels algorithmes multi-objectifs sera détaillé dans la partie 2.2.5. L'objectif ajouté pourra par exemple être la distance moyenne de l'individu considéré au reste de la population. En cherchant à le maximiser, on poussera donc la population à rester diverse. A noter que cette méthode introduit un objectif dynamique dans l'optimisation, étant donné que la valeur de diversité dépend de la population courante : une même solution peut avoir des valeurs de diversité différentes suivant la génération. En pratique, elle donne de bons résultats en permettant aux solutions de s'échapper des optima locaux très attractifs de l'espace de fitness [Mouret & Doncieux 2009b] ou dans le cas de problèmes de *bootstrap* [Mouret & Doncieux 2008, Mouret & Doncieux 2009a], c'est-à-dire quand la fonction de fitness ne fournit pas de gradient. Par ailleurs, certains travaux indiquent qu'optimiser un objectif de diversité est plus efficace qu'une technique de *fitness sharing* sur différents problèmes robotiques [Mouret & Doncieux 2011].

Une approche plus extrême part de l'hypothèse que la fonction de fitness est souvent trompeuse et ne permet généralement pas d'assurer un gradient utile vers les solutions optimales. En conséquence, il serait préférable de se contenter d'explorer l'espace des solutions sans être guidé par la fitness. On cherche alors à découvrir des solutions nouvelles se révélant parfois potentiellement intéressants pour la tâche à résoudre. Cette approche originale, appelée recherche par

nouveauté [Lehman & Stanley 2008, Lehman & Stanley 2010], repose sur deux ensembles de solutions, une population et une archive. Chaque solution de la population cherche à maximiser sa distance moyenne à ses n plus proches voisins dans la population et l'archive, et les solutions les plus nouvelles sont intégrées à l'archive. Appliquée à des tâches de navigation dans des environnements trompeurs (où l'utilisation d'une fonction de fitness simple échoue), l'approche permet de trouver des individus optimaux sans enjoindre directement les solutions à résoudre la tâche. En se focalisant sur l'exploration, l'approche peut cependant être longue à découvrir des solutions performantes, notamment si l'espace de recherche est vaste. Quelques travaux ont également montré qu'avec un même budget d'évaluations, il est parfois plus efficace d'optimiser conjointement fitness et diversité (ou fitness et nouveauté) avec un algorithme multi-objectifs qu'uniquement un objectif de nouveauté [Ollion & Doncieux 2011, Mouret 2011], la fonction de fitness permettant de guider la population vers des zones d'intérêt et d'éviter une exploration quasi-exhaustive de l'espace de recherche.

2.2 Optimisation multi-objectifs

Un problème d'optimisation met souvent en jeu plusieurs objectifs, contradictoires ou non [Fonseca & Fleming 1995, Deb 2001a]. Par exemple, on cherchera à maximiser la performance d'un processus, mais tout en minimisant son coût. Dans la suite de cette partie et afin d'illustrer les différentes méthodes et notions reliées à l'optimisation multi-objectifs, nous considérerons un problème d'optimisation visant à maximiser n objectifs. On note f_i la $i^{ème}$ fonction objectif du problème. Autrement dit, $f_i(\boldsymbol{x}) \in \mathbb{R}$ est la valeur du $i^{ème}$ objectif correspondant à la solution $\boldsymbol{x}$ et $\boldsymbol{f}(\boldsymbol{x})$ est le vecteur d'objectifs correspondant à $\boldsymbol{x}$. On se place dans le cas général où une solution $\boldsymbol{x}$ est un vecteur de m nombres réels et on appelle $\mathbb{X} \subset \mathbb{R}^m$ l'espace des solutions possibles ou encore espace de décision. Une fois toutes ces notations définies, on cherche donc à maximiser l'ensemble des fonctions objectifs :

$$\max_{\boldsymbol{x} \in \mathbb{X}} \quad f_i(\boldsymbol{x}), \quad i \in [1,\ n]$$

Il est important de distinguer l'espace de décision ou espace des solutions possibles qui est un sous-ensemble de $\mathbb{R}^m$ de l'espace des objectifs inclus dans $\mathbb{R}^n$: l'optimisation et la comparaison des solutions s'effectuent dans l'espace des objectifs, tandis que le choix final de l'utilisateur se fera dans l'espace de décision $\mathbb{X}$.

Comparé à un problème d'optimisation mono-objectif, il devient plus ardu de comparer les solutions. On utilise principalement la relation de dominance de Pareto pour effectuer les comparaisons :

- si une solution $\boldsymbol{x_1}$ est meilleure qu'une solution $\boldsymbol{x_2}$ sur tous les objectifs, on dit que $\boldsymbol{x_1}$ domine $\boldsymbol{x_2}$;
- si $\boldsymbol{x_1}$ est meilleure que $\boldsymbol{x_2}$ sur un sous-ensemble des objectifs, mais moins bonne sur les autres objectifs, aucune de ces deux solutions ne domine l'autre et on parle de solutions non-dominées.

Cette relation peut se formaliser comme suit. Soit deux solutions $\boldsymbol{x_1}$ et $\boldsymbol{x_2}$, la solution $\boldsymbol{x_1}$ domine la solution $\boldsymbol{x_2}$ si et seulement si :

$$\begin{cases} \forall i \in [1,\ n],\ f_i(\boldsymbol{x_1}) \geq f_i(\boldsymbol{x_2}) \\ \exists i \in [1,\ n],\ f_i(\boldsymbol{x_1}) > f_i(\boldsymbol{x_2}) \end{cases}$$

Une solution qui n'est dominée par aucune autre est dite Pareto-optimale et l'ensemble de toutes les solutions Pareto-optimales est appelé le front de Pareto. Comparé à un problème mono-objectif, on remarque donc qu'il est parfois impossible de comparer des solutions : deux solutions non-dominées sont considérées comme des compromis équivalents dans l'espace des objectifs.

De nombreuses méthodes d'optimisation existent pour gérer les problèmes multi-objectifs [Coello 1999]. On peut distinguer principalement deux familles : (MO1) les méthodes cherchant une unique solution appartenant au front de Pareto ; (MO2) les méthodes cherchant un ensemble de solutions non-dominées qui approchent au mieux le front de Pareto.

2.2.1 Rechercher une unique solution Pareto-optimale (MO1)

Les méthodes de la famille (MO1) peuvent être réparties en trois principaux groupes [Coello 1999] suivant qu'elles soient basées sur :

- MO1-a. l'agrégation des objectifs en une unique fonction à optimiser [Wilson & Macleod 1993, Yang & Gen 1994, Liu *et al.* 1998].
- MO1-b. la prise en compte de certains objectifs en tant que contraintes [Ranjithan *et al.* 1992, Quagliarella & Vicini 1997].
- MO1-c. la recherche d'une solution-cible dans l'espace des objectifs [Charnes & Cooper 1957, Ijiri 1965, Duckstein 1981].

2.2.1.1 Agréger les objectifs

Agréger les objectifs en une unique fonction de performance (MO1-a) peut être réalisé par différents moyens, que ce soit en utilisant une combinaison linéaire des objectifs (i.e. une somme pondérée), en ayant recours à une fonction de distance par rapport à un point idéal, ou encore en construisant une agrégation plus complexe sous forme d'une fonction de valeur. Quelle que soit la méthode utilisée, le but est le même : se ramener à un problème d'optimisation mono-objectif classique.

Nous nous étendrons principalement sur l'utilisation de combinaisons linéaires, une des méthodes les plus simples et répandues pour agréger plusieurs objectifs [Wilson & Macleod 1993, Yang & Gen 1994, Liu *et al.* 1998]. En supposant qu'un vecteur de poids $\boldsymbol{w}$ de taille n a été fourni par l'utilisateur, le problème d'optimisation est reformulé de la façon suivante :

$$\max_{\boldsymbol{x} \in \mathbb{X}} \quad \sum_{i=1}^{n} w_i\ f_i(\boldsymbol{x}), \quad i \in [1,\ n]$$

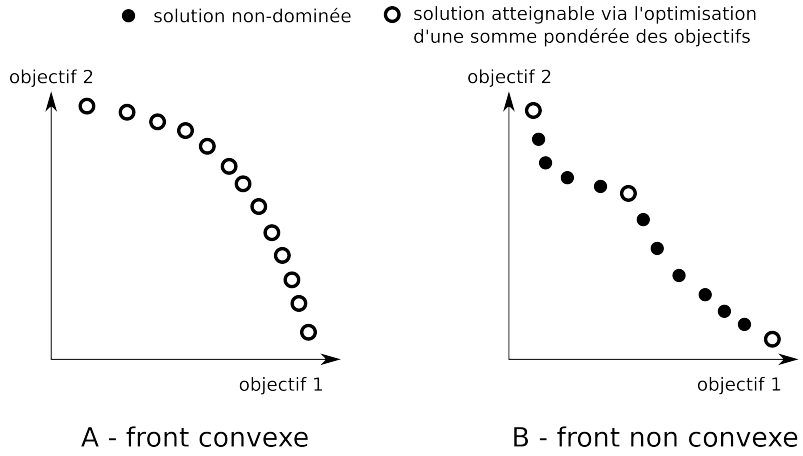

FIGURE 2.3 – Exemple de fronts de Pareto convexe et non convexe dans un problème de maximisation de deux objectifs — Indication des solutions pouvant être obtenues à l'aide d'une combinaison linéaire des objectifs.

L'efficacité d'une telle méthode est fortement dépendante de la forme du front de Pareto correspondant au problème. En effet, si le front est convexe comme sur la figure 2.3-A, toute solution du front peut être obtenue en maximisant une combinaison linéaire des objectifs. Dans le cas de la maximisation de deux objectifs, la solution non-dominée obtenue est l'intersection entre le front de Pareto et la droite dont l'ordonnée à l'origine est maximale parmi les droites de coefficient directeur $-\frac{w_1}{w_2}$ et d'intersection non nulle avec le front. Comme la solution obtenue dépend du choix des valeurs de poids, il est alors théoriquement possible de construire le front de Pareto en effectuant une série d'optimisations mono-objectif avec différents vecteurs de poids. Par contre, dans le cas où le front de Pareto n'est pas convexe, optimiser une combinaison linéaire peut exclure de nombreuses zones du front (cf. figure 2.3-B).

En pratique, comme il est difficile de connaître à l'avance la forme du front de Pareto, l'efficacité de méthodes basée sur l'agrégation des objectifs est très variable d'un problème à l'autre.

Une alternative à l'utilisation de combinaisons linéaires des objectifs est de chercher à minimiser la distance des solutions au point idéal. Le point idéal $\boldsymbol{f}^*$ est défini comme le point de l'espace des objectifs qui a la valeur maximale possible sur chaque objectif :

$$\forall i \in [1,\ n] \quad f_i^* = \max_{\boldsymbol{x}\ \in\ \mathbb{X}} f_i(\boldsymbol{x})$$

En supposant que ce point est connu et qu'une fonction de distance d et un vecteur de poids $\boldsymbol{w}$a été fournie par l'utilisateur, le problème d'optimisation multi-objectifs se formule comme suit :

$$\min_{\boldsymbol{x} \in \mathbb{X}} \quad (\sum_{i=1}^{n} w_i \ |f_i(\boldsymbol{x}) - f_i^*|^p)^{\frac{1}{p}}$$

p est un paramètre réel fixé par l'utilisateur dans l'intervalle $[1, +\infty]$. Si la valeur de p est bien choisie, cette méthode peut potentiellement trouver toute solution Pareto-optimale en faisant varier le vecteur de poids $\boldsymbol{w}$. Il est néanmoins nécessaire de connaître le point idéal $\boldsymbol{f^*}$, par exemple en lançant au préalable n optimisations mono-objectif pour connaître les valeurs maximales atteignables par chaque objectif. Cette étape peut être très coûteuse en temps de calcul.

Une dernière façon d'agréger les objectifs revient à construire une fonction d'utilité définie par l'utilisateur sur l'ensemble de l'espace des objectifs. Le problème d'optimisation revient ensuite à maximiser cette fonction. On peut considérer qu'il s'agit d'une généralisation des méthodes précédentes, étant donné qu'elle autorise toute relation linéaire ou non-linéaire combinant les objectifs. Néanmoins, le choix de la fonction est difficile et contraint évidemment la solution finale de l'optimisation.

2.2.2 Méthode par ε-contraintes

Les méthodes de la famille (MO1-b) distinguent un des objectifs qui sera effectivement maximisé des autres qui seront gérés en tant que contraintes [Ranjithan *et al.* 1992, Quagliarella & Vicini 1997]. L'idée est de définir des bornes inférieures ε sur les $n-1$ objectifs utilisés comme contraintes. On cherche ensuite la solution qui maximise le dernier objectif dans le sous-espace de l'espace des objectifs ainsi défini. En supposant que la fonction objectif f_1 soit maximisée avec ε_i la borne inférieure définie pour la fonction objectif f_i, le problème se formule de la façon suivante :

$$\begin{array}{ll} \max\limits_{\boldsymbol{x} \in \mathbb{X}} & f_1(\boldsymbol{x}) \\ s.c.q.\,[3] & f_i \geq \varepsilon_i \quad i \in [2,\ n] \end{array}$$

En faisant varier les bornes inférieures ε_i, il est possible d'atteindre différentes solutions Pareto-optimales, indépendamment des caractéristiques du front de Pareto. Néanmoins cette méthode impose à l'utilisateur le choix de bornes généralement inter-dépendantes dans un espace de dimensions $n-1$. Poser de mauvaises bornes sur les objectifs peut de plus contraindre la recherche de solutions en-dehors des parties de l'espace de décision compatibles avec les contraintes et, dans ce cas, la méthode ne trouvera pas de solution.

2.2.3 Atteindre un point-cible

La dernière méthode que nous aborderons pour la famille (MO1) repose sur la définition d'un point spécial $\boldsymbol{f}_{but}$ dans l'espace des objectifs [Charnes & Cooper 1957,

3. s.c.q. = sous contrainte que

Ijiri 1965, Duckstein 1981]. On cherche alors la solution $\boldsymbol{x}$ correspondant à des valeurs d'objectifs égales à celles de ce point-cible. Comme il est parfois impossible de satisfaire exactement une telle contrainte, des déviations sont autorisées autour du point-cible et on cherche alors à minimiser ces déviations. Soit $\boldsymbol{\delta}(\boldsymbol{x})$ le vecteur de déviations entre le but $\boldsymbol{f}_{but}$ et $\boldsymbol{f}(\boldsymbol{x})$ dans l'espace des objectifs, on cherche donc :

$$\begin{array}{lll} \min\limits_{\boldsymbol{x} \in \mathbb{X}} & |\delta_i(\boldsymbol{x})| & i \in [1,\ n] \\ s.c.q. & \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\delta}(\boldsymbol{x}) = \boldsymbol{f}_{but} & i \in [2,\ n] \end{array}$$

Le problème d'optimisation à résoudre est toujours multi-objectifs, bien qu'il porte maintenant sur les déviations par rapport au point-cible. Différentes méthodes existent pour minimiser les déviations, par exemple en minimisant une somme pondérée des déviations ou encore le maximum sur les déviations. Là encore, l'utilisateur doit apporter de l'information sur le problème en choisissant le point-cible à atteindre.

2.2.4 MO1 : Conclusions

Toutes les méthodes présentées ci-dessus visent à trouver, pour un problème d'optimisation multi-objectifs, une solution Pareto-optimale en se ramenant à une optimisation mono-objectif. Elles se distinguent par la façon dont l'information des différents objectifs est gérée, mais il est à noter que l'utilisateur est souvent partie intégrante du processus, ce qui rend ce type d'optimisation difficilement automatisable. D'autre part, l'utilisateur peut souhaiter obtenir plusieurs solutions Pareto-optimales avant de choisir un bon compromis dans l'espace de décision. Il est alors nécessaire de lancer plusieurs optimisations mono-objectif, ce qui n'est pas envisageable si l'évaluation des solutions est lente ou coûteuse.

2.2.5 Utiliser des algorithmes évolutionnistes - (MO2)

Les méthodes de la famille (MO2) visent à trouver un ensemble de solutions non-dominées proches du front de Pareto du problème et donc de construire une approximation du front dans l'espace des objectifs au lieu de chercher les solutions une par une. L'utilisation des algorithmes évolutionnistes dans ce cadre est assez directe pour deux raisons :

- ils optimisent une population de solutions ;
- la prise en compte des objectifs est directement intégrable lors du classement des solutions avant l'étape de sélection.

L'idée principale derrière l'utilisation des algorithmes évolutionnistes est que la population permet de construire directement une approximation du front de Pareto du problème à résoudre en une seule optimisation. La qualité de l'approximation va dépendre de deux principaux facteurs :

- la convergence des solutions optimisées vers le front de Pareto ;
- leur dispersion tout au long du front.

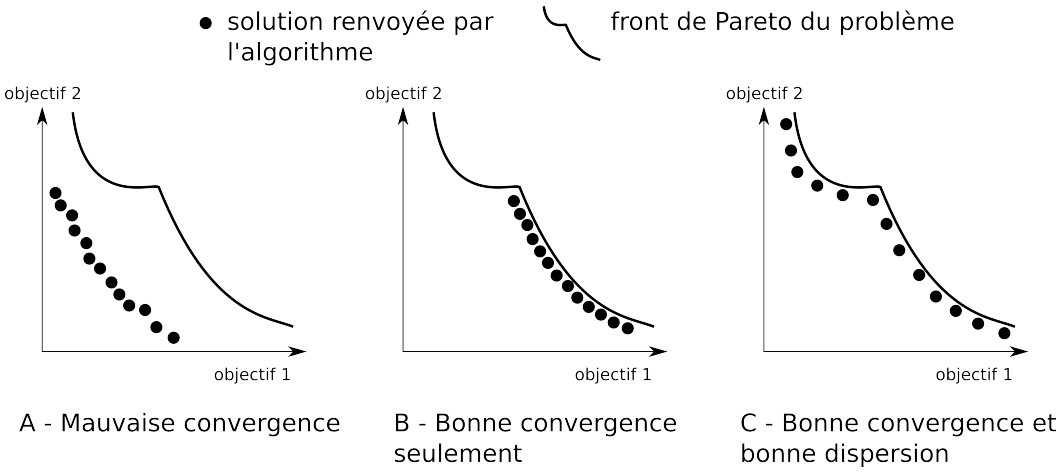

FIGURE 2.4 – Exemples fictifs d'approximation d'un front de Pareto par un algorithme évolutionniste dans un problème visant à maximiser deux objectifs.

La figure 2.4 illustre le besoin de ces deux facteurs. En effet, assurer une bonne dispersion des solutions dans l'espace des objectifs n'a pas de sens si les solutions obtenues sont loin du front de Pareto. De même, rechercher juste la convergence vers le front n'est pas non plus efficace, car les solutions obtenues risquent de se concentrer dans une partie très localisée du front. En conséquence, lors de la sélection des meilleures solutions, il est souvent nécessaire de prendre en compte des facteurs de dispersion dans l'espace des objectifs et pas seulement la relation de dominance entre individus.

Suivant l'application considérée, un processus de décision peut avoir lieu sur l'approximation du front de Pareto obtenu une fois l'optimisation terminée : l'utilisateur choisit alors une ou plusieurs solutions qui présentent de bons compromis entre les objectifs. Cette étape est très dépendante du problème à résoudre, ce qui rend son automatisation difficile (i.e. définir un critère numérique systématiquement maximisé/minimisé par le(s) "meilleur(s)" compromis). Il est parfois même nécessaire de laisser l'utilisateur choisir la solution lui-même afin d'éviter une mauvaise sélection [Lucet *et al.* 2010].

2.2.6 Exemples d'algorithmes

Les algorithmes évolutionnistes multi-objectifs de l'état de l'art se distinguent principalement par leur gestion de la balance entre convergence et dispersion lors du classement et de la sélection des solutions. Comme notre ne dépend pas crucialement du choix d'un algorithme multi-objectif donné, nous avons fait le choix de nous focaliser sur quatre algorithmes classiques (NSGA-II étant sans doute le plus utilisé à l'heure actuelle), ceci dans le but d'illustrer différentes stratégies de sélection permettant d'assurer à la fois la convergence des solutions vers le front de Pareto et leur dispersion le long du front (pour une revue récente sur les algorithmes

évolutionnistes multi-objectif, voir [Coello *et al.* 2007]) :

- Strength Pareto Evolutionary Algorithm (SPEA) [Zitzler & Thiele 1998a] ;
- Non-dominated Sorting Genetic Algorithm II (NSGA-II) [Deb *et al.* 2000, Deb *et al.* 2002b] ;
- ε-Multiobjective Evolutionary Algorithm (ε-MOEA) [Deb *et al.* 2005] ;
- les algorithmes basés sur la mesure d'hypervolume [Fleischer 2003].

2.2.6.1 Strength Pareto Evolutionary Algorithm, 1998

L'algorithme SPEA a été proposé en 1998 [Zitzler & Thiele 1998a]. En plus de la population de solutions optimisées, il est basé sur l'utilisation d'une archive qui engrange les solutions non-dominées au fur et à mesure qu'elles apparaissent dans la population. La fitness d'un individu de l'archive est calculée en fonction du nombre de solutions qu'il domine dans la population. Ensuite, la fitness d'un individu de la population est la somme des fitness des individus de l'archive qui le dominent.

Considérons l'exemple fictif de la figure 2.5 avec une population de 5 individus et une archive contenant 2 solutions non-dominées. L'individu p5 étant non-dominé que ce soit dans la population ou dans l'archive, il est ajouté à l'archive (cf. figure 2.5-B). Les individus de l'archive sont ensuite évalués (cf. figure 2.5-C) : par exemple, comme la solution a1 domine deux individus de la population, p1 et p3, on lui assigne une fitness de 2 divisé par la taille de la population plus 1, soit $\frac{2}{6} \simeq 0.33$. De manière similaire, a2 domine p3 et p4 et reçoit une fitness de $\frac{2}{6} \simeq 0.33$. Par contre, a3 domine trois individus, p2, p3 et p4, et on lui assigne donc une fitness de $\frac{3}{6} = 0.5$.

Une fois toutes les solutions de l'archive évaluées, les individus de la population le sont à leur tour (cf. figure 2.5-D). Par exemple, comme p1 est uniquement dominé par a1, il reçoit comme fitness la valeur de fitness de a1 plus 1, c'est-à-dire $1+0.33 = 1.33$. De même, p2 est dominé par a3, donc il reçoit $1 + 0.5 = 1.5$. Suivant la même méthode, p3 est dominé par a2 et a3, donc il reçoit $1 + 0.33 + 0.5 = 1.83$ et p4 qui est dominé par a1, a2 et a3, reçoit $1 + 0.33 + 0.33 + 0.5 = 2.16$.

Moins la valeur de fitness d'un individu est élevée, plus il sera sélectionné, ce qui entraîne deux effets principaux :

- les individus non-dominés sont conservés car ils ont toujours des fitness plus basses que les individus dominés ;
- les solutions qui dominent ou sont dominés par peu d'individus sont privilégiées, étant placées dans des zones potentiellement intéressantes au regard de la dispersion des solutions.

A terme, l'algorithme renvoie un ensemble de solutions non-dominées avec une bonne dispersion et une densité relativement homogène le long du front de Pareto.

Comme la taille de l'archive peut rapidement augmenter, notamment dans le cas de objectifs à valeurs continues réelles, l'algorithme introduit également une taille maximale fixe N au-delà de laquelle des individus non-dominés doivent être enlevés de l'archive. Cette étape est assurée par une méthode de *clustering* qui vise à garder les solutions les plus représentatives présentes dans l'archive. Sans entrer dans le détail de la méthode utilisée, si la taille de l'archive dépasse le seuil N, les

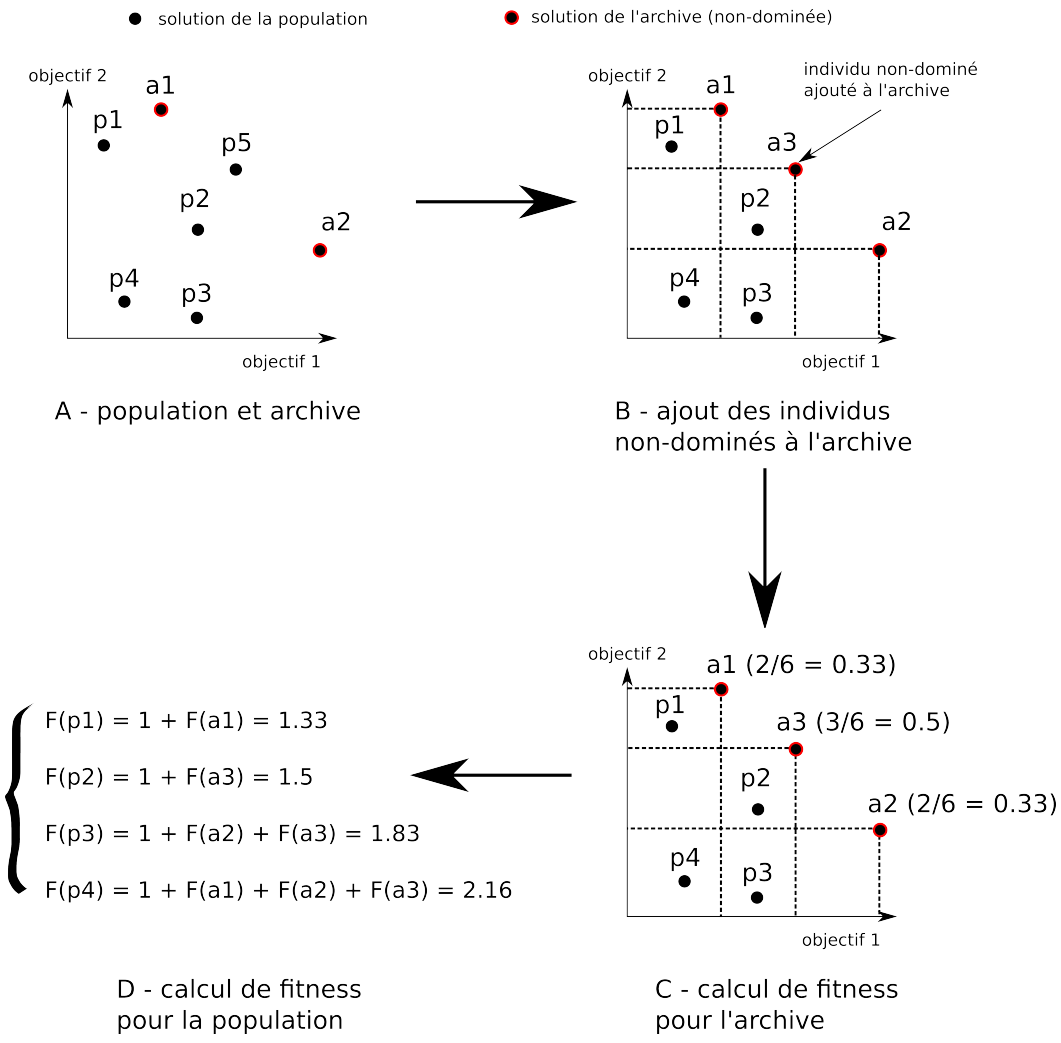

FIGURE 2.5 – Calcul des fitness pour l'algorithme SPEA basé sur la relation de dominance dans un problème fictif visant à maximiser deux objectifs.

solutions sont réparties en N *clusters* dans l'espace des objectifs et on ne conserve que le centroïde de chaque *cluster* dans l'archive, c'est-à-dire le point dont la distance moyenne à tous les autres points du *cluster* est minimale. Ceci garantit de maintenir la dispersion des solutions dans l'espace des objectifs tout en évitant les problèmes posés par une archive de trop grande taille.

Néanmoins, l'algorithme de *clustering* utilisé dans SPEA a une complexité élevée, ce qui augmente d'autant la complexité globale de l'algorithme. D'autre part, il s'avère que la taille maximale autorisée pour l'archive peut avoir des effets néfastes sur la convergence de l'algorithme si l'utilisateur la choisit trop grande ou trop basse [Deb 2001a]. Une version améliorée de cet algorithme appelée SPEA-II et utilisant notamment un algorithme de *clustering* de plus faible complexité a été proposée dans l'article [Zitzler *et al.* 2001].

2.2.6.2 Non-dominated Sorting Genetic Algorithm II, 2000

L'algorithme NSGA-II a été proposé en 2000 [Deb *et al.* 2000, Deb *et al.* 2002b] en tant qu'algorithme d'optimisation multi-objectifs rapide. En effet, un algorithme évolutionniste multi-objectifs ne doit pas seulement assurer une bonne convergence vers le front de Pareto et une bonne dispersion des solutions dans l'espace des objectifs, mais doit aussi s'exécuter rapidement. Pour ce faire, NSGA-II introduit un classement des individus en plusieurs fronts successifs suivant la relation de dominance et propose également de sélectionner les solutions en fonction d'un critère de performance et de la densité de solutions dans leur voisinage.

Le principe est le suivant. Considérant une population d'individus et leurs valeurs respectives d'objectifs, il est possible de les répartir dans des fronts successifs (figure 2.6). Les individus non-dominés sont dans le premier front. Le deuxième front est composé des individus non-dominés une fois les individus du premier front retirés de la population et ainsi de suite jusqu'à ce que tous les individus soient assigné à un front. Une fois cette étape effectuée, la fitness d'un individu est le numéro du front dans lequel il a été affecté. En particulier, tous les individus initialement non-dominés reçoivent une fitness de 1.

De plus, chaque solution se voit assigner une distance de *crowding* qui indique, à l'intérieur d'un front donné, si la solution est dans une région plutôt dense ou clairsemée. Considérant le $j^{ème}$ front défini précédemment, on classe tout d'abord les solutions de ce front par valeurs croissantes sur le premier objectif. Soit le $i^{ème}$ individu du front classé, sa valeur de distance de *crowding* est calculée comme la distance de Manhattan entre l'individu $i-1$ et $i+1$. La figure 2.6-B illustre le calcul en 2 dimensions. Si la solution est à une extrémité du front, sa distance de *crowding* est fixée à une valeur arbitrairement élevée afin que les solutions extrêmes aient toujours la distance de *crowding* maximale dans le front considéré.

A chaque solution x correspond donc un couple de valeurs $(i,\ d)$, où i est le numéro de son front et d la valeur de sa distance de *crowding*. Soient deux individus $\boldsymbol{x_1}$ et $\boldsymbol{x_2}$ correspondant respectivement aux couples $(i_1,\ d_1)$ et $(i2,\ d_2)$, $\boldsymbol{x_1}$ sera préféré à $\boldsymbol{x_2}$ si l'une des deux conditions suivantes est vérifiée :

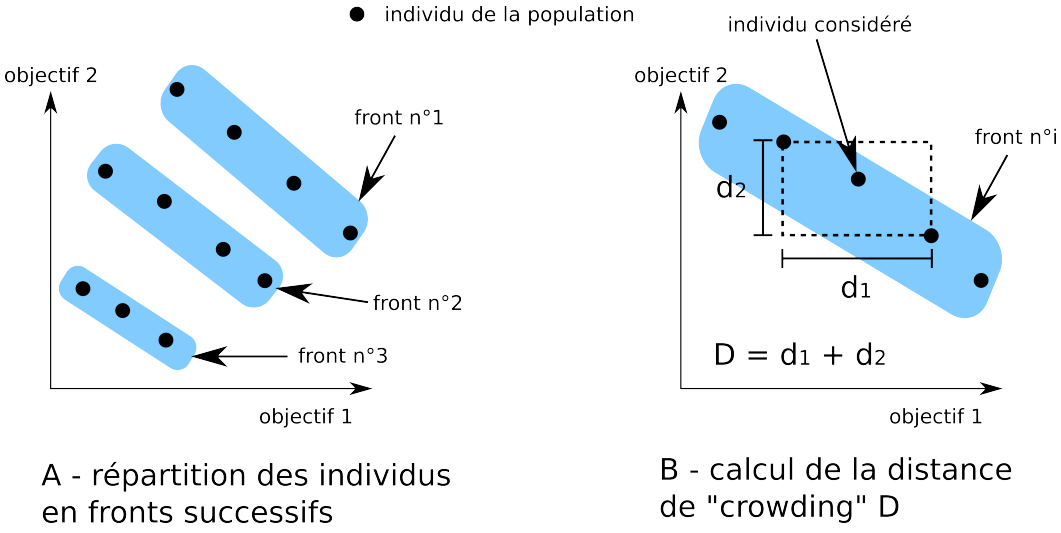

FIGURE 2.6 – Illustration de la procédure de classement des individus dans l'algorithme NSGA-II dans un problème fictif où deux objectifs sont maximisés.

$$\begin{cases} i_1 < i_2 \\ \text{ou} \\ (i_1 = i_2) \text{ et } (d_1 > d_2) \end{cases}$$

Dans le premier cas, $\boldsymbol{x_1}$ fait partie d'un front qui contient au moins une solution dominant $\boldsymbol{x_2}$. Dans le deuxième cas, $\boldsymbol{x_1}$ et $\boldsymbol{x_2}$ sont dans un même front et on préfère $\boldsymbol{x_1}$ qui a la distance de *crowding* la plus élevée, c'est-à-dire la solution située dans la région la moins dense. Ainsi, des solutions d'un même front qui sont donc non-dominées entre elles et équivalentes d'après leurs valeurs d'objectifs, ne sont différenciées que suivant leur dispersion.

Cette méthode de sélection a un coût computationnel moins élevé que le *clustering* mis en jeu avec l'algorithme SPEA. De nombreuses comparaisons entre SPEA et NSGA-II ont été menées [Zitzler *et al.* 2001, Deb 2001a, Deb *et al.* 2002b, Deb *et al.* 2005], indiquant généralement que les deux algorithmes obtiennent des résultats de qualité similaire en terme de convergence et de dispersion des solutions, malgré le gain significatif en temps de calcul obtenu avec NSGA-II. Une variante améliorée de NSGA-II a également été proposée, utilisant un algorithme de tri des solutions non-dominées plus rapide et menant à des complexités algorithmes très intéressantes pour un algorithme multi-objectifs [Jensen 2003].

2.2.6.3 ε-Multiobjective Evolutionary Algorithm, 2005

Le troisième et dernier algorithme évolutionniste multi-objectifs auquel nous allons nous intéresser ici repose sur une version modifiée de la relation de dominance : l'ε-dominance [Laumanns *et al.* 2002, Groşan & Oltean 2004, Deb *et al.* 2005]. Soit

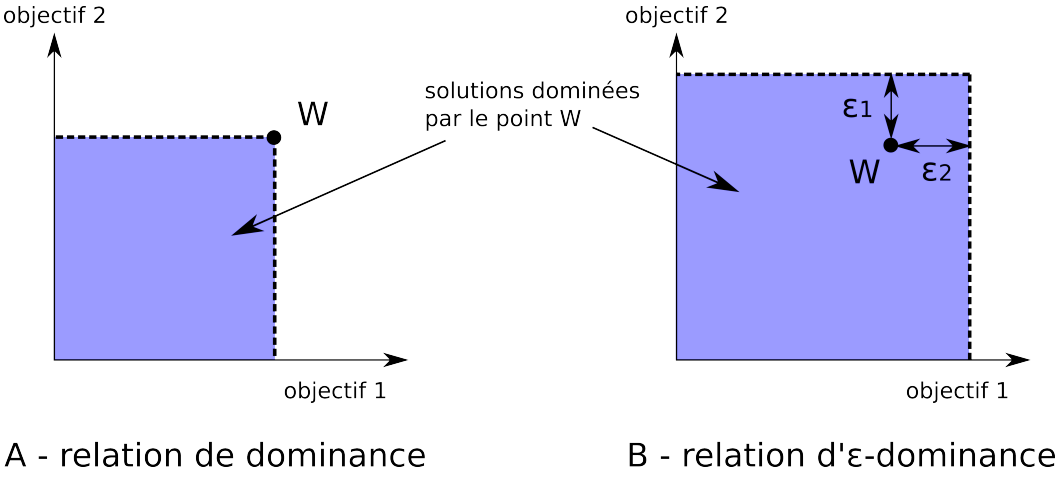

FIGURE 2.7 – Illustration des différences entre la relation de dominance et la relation d'ε-dominance.

$\boldsymbol{\varepsilon}$ un vecteur quelconque de n réels strictement positifs, n étant le nombre de objectifs optimisés, on dit que la solution $\boldsymbol{x_1}$ ε-domine la solution $\boldsymbol{x_2}$ si :

$$\begin{cases} \forall i \in [1,\ n],\ f_i(\boldsymbol{x_1}) + \varepsilon_i \geq f_i(\boldsymbol{x_2}) \\ \exists i \in [1,\ n],\ f_i(\boldsymbol{x_1}) + \varepsilon_i > f_i(\boldsymbol{x_2}) \end{cases}$$

La différence entre la relation de dominance habituelle et l'ε-dominance est illustrée sur la figure 2.7 : avec l'ε-dominance, une solution domine toutes les solutions qui ne sont pas suffisamment meilleures qu'elle sur au moins un objectif, c'est-à-dire si la différence sur un objectif ne dépasse pas la valeur ε_i correspondante. A noter que le front de Pareto ε-approché défini par la relation d'ε-dominance n'est pas unique.

La relation d'ε-dominance possède de bonnes propriétés théoriques (voir par exemple [Laumanns *et al.* 2002]). Elle assure notamment une bonne convergence et une bonne dispersion des solutions du front de Pareto ε-approché vis-à-vis du front de Pareto "exact" défini par la relation de dominance. Le front de Pareto ε-approché a également une taille bornée en fonction des valeurs d'ε utilisées.

L'algorithme ε-MOEA [Laumanns *et al.* 2002, Deb *et al.* 2005] utilise cette relation d'ε-dominance plus que la relation de dominance classique. Il repose sur deux ensembles d'individus, une population et une archive contenant les solutions non-dominées trouvées. A chaque étape, deux parents sont sélectionnés, un dans la population et un dans l'archive, et sont utilisés pour générer deux enfants. Ces nouveaux individus sont ensuite intégrés à la population selon la relation classique de dominance et ajoutés à l'archive selon la relation d'ε-dominance. L'idée principale est d'assurer une bonne dispersion des solutions non-dominées dans l'espace des objectifs, en ne gardant que quelques solutions représentatives le long du front dans l'archive. Suivant les valeurs ε_i fixées pour chaque objectif, l'utilisateur peut ainsi

intégrer de l'information dans le processus d'optimisation en indiquant l'écart sur chaque objectif à partir duquel deux solutions seront effectivement différentes de son point de vue [Laumanns *et al.* 2002].

L'algorithme ε-MOEA permet ainsi d'obtenir de bonnes performances en terme de convergence et de dispersion des solutions tout en ayant un coût computationnel moins élevé que SPEA et NSGA-II. Néanmoins, le choix des valeurs ε_i a une influence évidente sur l'approximation finale du front de Pareto. D'autre part, la densité de solutions peut également être variable le long du front et l'utilisation d'une constante ε_i par objectif risque d'entraîner un sous-échantillonnage de certaines zones par rapport à d'autres.

2.2.6.4 Algorithmes basés sur la mesure d'hypervolume

La mesure d'hypervolume (aussi appelée métrique S) a d'abord été utilisée comme critère permettant de comparer les performances d'algorithmes multi-objectifs [Zitzler & Thiele 1998b, Laumanns *et al.* 1999]. Cette mesure correspond au volume de la partie de l'espace des objectifs dominé par un ensemble de solutions donné et bornée par un point de référence arbitraire z_{ref} (illustration dans le cas de deux objectifs sur la figure 2.8). Il a ensuite été prouvé théoriquement par Fleischer [Fleischer 2003] que maximiser la mesure d'hypervolume est une condition nécessaire et suffisante pour obtenir à la fois la convergence des solutions vers le front de Pareto ainsi qu'une dispersion maximale des solutions le long du front, et ce indépendemment du nombre d'objectifs. L'utilisation de cette mesure permet donc théoriquement de transformer un problème d'optimisation multi-objectif en un problème de maximisation d'un unique scalaire, l'hypervolume [Fleischer 2003].

Le calcul exact de cette mesure est d'une complexité importante, ce qui empêche en pratique son utilisation directe sur de grandes populations ou avec de nombreux objectifs : la complexité du calcul de l'hypervolume sur un ensemble de k individus avec n objectifs à optimiser est estimée à $O(k^{n+1})$ d'après Knowles et Corne [Knowles & Corne 2003]. Plusieurs travaux se sont depuis intéressés à réduire la complexité du calcul de l'hypervolume, que ce soit pour obtenir une estimation exacte (par exemple avec l'algorithme HSO [While *et al.* 2006]) ou via une heuristique (par exemple une méthode Monte-Carlo [Bader & Zitzler 2011] ou en considérant un sous-ensemble des objectifs [Brockhoff & Zitzler 2007]).

D'autre part, la définition du point de référence z_{ref} peut avoir une influence importante sur les valeurs d'hypervolume et la préférence d'un ensemble de solutions non-dominées à un autre peut être variable suivant le point de référence défini [Knowles & Corne 2002, Knowles & Corne 2003].

Bien que l'utilisation de la mesure d'hypervolume semble une voie prometteuse pour traiter les problèmes d'optimisation multi-objectif, il existe actuellement peu d'algorithmes établis l'utilisant directement. La plupart des algorithmes basés sur cette mesure (par exemple MO-CMA-ES de Igel et al. [Igel *et al.* 2007], SMS-EMOA de Beume et al. [Beume *et al.* 2007] ou encore HypE de Bader et Zitzler [Bader & Zitzler 2011]) l'utilise en effet plutôt comme une alternative à la dis-

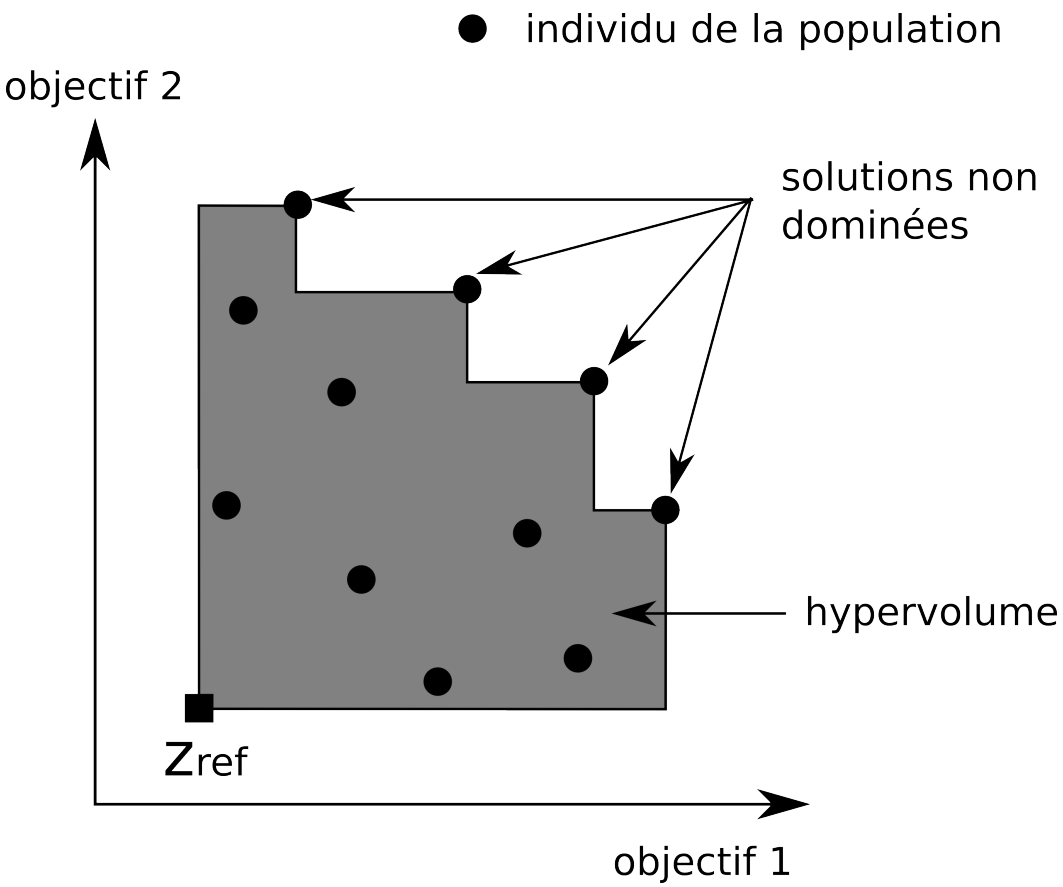

FIGURE 2.8 – Illustration du calcul de l'hypervolume sur un problème de maximisation de deux objectifs pour un point de référence z_{ref} donné.

tance de "crowding" dans le cadre d'un tri des solutions non-dominées inspiré de l'algorithme NSGA-II (se reporter à la section 2.2.6.2 pour plus de détails), c'est-à-dire comme un critère utilisé pour distinguer les solutions appartenant à un même front.

2.2.7 Limitations et conclusions

Bien que les algorithmes évolutionnistes soient des outils très pratiques pour l'optimisation multi-objectifs, ils présentent deux limitation majeures. En premier lieu, les algorithmes actuels fonctionnent mal au-delà de trois ou quatre objectifs antagonistes [Deb *et al.* 2002c, Ishibuchi & Nojima 2006, Khare *et al.* 2003, Teytaud 2007]. Il a notamment été montré qu'à nombre d'évaluations égal, NSGA-II est moins efficace qu'une série d'optimisation multi-objectifs en optimisant 4 ou 6 objectifs antagonistes, alors qu'il est clairement meilleur avec 2 objectifs [Hughes 2005]. En effet, comme le nombre de solutions Pareto-optimales croît exponentiellement en fonction du nombre d'objectifs, la taille de population nécessaire pour obtenir une

approximation du front de qualité constante croît également. En pratique, quand la taille de population est trop faible, l'ensemble des solutions optimisées se retrouve rapidement non-dominées, ce qui réduit drastiquement l'efficacité de la phase de sélection : les solutions sont *a priori* toutes équivalentes. Même si quelques méthodes sont proposées pour dépasser cette limitation [Ishibuchi *et al.* 2008], il n'existe pas à l'heure actuelle de méthode évolutionniste facilement applicable quel que soit le nombre d'objectifs et dans un temps raisonnable. Certains résultats théoriques tendent même à montrer qu'au-delà de 3 objectifs antagonistes, le taux de convergence de la plupart des algorithmes multi-objectifs est assimilable à celui d'une recherche aléatoire [Teytaud 2007].

Le deuxième problème qui a également été soulevé pour les méthodes de la famille (MO1) intervient si l'évaluation des solutions est lente ou coûteuse. Dans ce cas, le fait d'optimiser une population de solutions peut s'avérer handicapant.

Malgré ces limitations, les algorithmes évolutionnistes sont une solution élégante au problème de l'optimisation multi-objectifs, bien que les algorithmes actuels ne soient efficaces que jusqu'à 4 objectifs antagonistes. Au-delà, les performances dépendent essentiellement de l'antagonisme entre les objectifs : plus les objectifs sont antagonistes entre eux, plus la taille du front de Pareto risque d'augmenter et moins l'algorithme sera performant.

Dans les différents travaux que nous présenterons au cours de ce manuscrit, nous nous avons privilégié l'utilisation de l'algorithme NSGA-II, sans doute l'un des algorithmes évolutionnistes multi-objectif les plus utilisés actuellement [Coello & Lamont 2004, Coello Coello 2006] dans des domaines variés (par exemple en chimie [Nandasana *et al.* 2003], en médecine [Lahanas *et al.* 2003, Lahanas *et al.* 2004], en écologie [Bekele & Nicklow 2007] ou encore en ingénierie [Ray 2004, Reed *et al.* 2007]), étant donné qu'il est à la fois simple à utiliser, rapide et compatible avec l'utilisation d'objectifs dynamiques tels que des objectifs de diversité [Mouret & Doncieux 2011] dont nous ferons usage (voir section 2.1.4).

2.2.8 Optimisation multi-objectifs et gestion de contraintes

Un grand nombre de problèmes d'optimisation mettent en jeu des contraintes sur l'espace de décision, afin de ne considérer que le sous-ensemble de solutions potentielles ayant un sens pour le problème considéré. Les contraintes peuvent être soit des égalités, soit des inégalités, et s'appliquent sur des fonctions des variables de décision. Un individu qui ne vérifie pas une des contraintes du problème est dite irréalisable et le but du processus d'optimisation est donc de trouver une solution à la fois performante et réalisable. On peut distinguer les contraintes dures, qui doivent absolument être satisfaites pour que la solution soit recevable, des contraintes molles pouvant être relaxées.

Il existe de nombreuses méthodes pour gérer des contraintes. Une des plus simples revient à exclure les solutions irréalisables et n'effectuer l'optimisation qu'avec celles vérifiant les contraintes. Comme il peut être difficile de trouver une solution initiale réalisable, certains travaux proposent des méthodes plus flexibles basées sur un sys-

tème de pénalité [Richardson *et al.* 1989, Coello 2000b] : pour chaque contrainte violée, une valeur arbitraire fixée par l'utilisateur est retranchée à la valeur de l'objectif maximisé. Dans le cas de nombreuses contraintes, l'utilisateur devra donc fixer autant de valeurs de paramètres.

Une alternative prometteuse est le recours aux algorithmes évolutionnistes multi-objectifs [Fonseca & Fleming 1998, Deb 2001a, Coello 2000a, Coello *et al.* 2007]. Chaque contrainte peut être gérée indépendamment sous la forme d'un objectif. Une fois l'approximation du front de Pareto obtenue, il est alors possible de considérer le sous-ensemble des solutions qui vérifient les contraintes, puis de choisir la solution réalisable qui maximise l'objectif initialement optimisé. Contrairement à l'utilisation de fonctions de pénalité, cette méthode a l'avantage de ne pas introduire de paramètres supplémentaires. Par contre, si les solutions réalisables sont très localisées sur le front de Pareto, le fait de construire une approximation de l'ensemble du front peut se révéler inefficace : seule la partie extrême du front correspondant à la zone où toutes les contraintes sont satisfaites va intéresser l'utilisateur à terme [Deb 2001a].

D'autre part, si la tâche compte de nombreuses contraintes, le nombre d'objectifs est alors trop grand pour être géré efficacement par un algorithme évolutionniste. Une alternative consiste à prendre en compte le nombre de contraintes satisfaites par chaque solution ou la somme de termes de violation pour toutes les contraintes [Coello 2000a]. L'utilisation des algorithmes évolutionnistes multi-objectifs dans le cadre de l'optimisation sous contraintes semble donc intéressante quand le nombre de contraintes est peu élevé et notamment dans le cas de contraintes molles.

2.3 La robotique évolutionniste

La robotique évolutionniste [Davidor 1991, Cliff *et al.* 1993b, Harvey *et al.* 1993, Meyer *et al.* 1998, Nolfi & Floreano 2000] regroupe les travaux utilisant les algorithmes évolutionnistes pour concevoir de manière automatique des contrôleurs ou des morphologies pour des robots. L'un des buts premiers de ce domaine est de construire automatiquement des contrôleurs pour des robots autonomes qui résolvent des tâches utiles dans des environnements inconnus ou non maîtrisés [Brooks 1992, Cliff *et al.* 1993b, Meyer *et al.* 1998]. Contrairement aux méthodes issues de la théorie du contrôle où le contrôleur est directement dérivé d'un modèle mathématique de la dynamique du robot, les algorithmes évolutionnistes se basent uniquement sur une valeur de performance obtenue pour chaque contrôleur en effectuant une expérience sur le robot ou en simulation. Comme les contrôleurs peuvent alors potentiellement exploiter toutes les spécificités du robot, aussi bien sensorielles que motrices, les algorithmes évolutionnistes sont utilisés pour générer des solutions simples et originales à des problèmes robotiques sans intervention du concepteur ou de l'utilisateur [Cliff *et al.* 1993b, Meyer *et al.* 1998]. Nous nous intéresserons principalement dans ce manuscrit à la conception automatique de contrôleurs.

Dans une expérience typique de robotique évolutionniste, la performance

d'un contrôleur est calculée via une expérience sur le robot ou en simulation. Certains travaux se placent alors dans un contexte d'optimisation boîte noire : sans *a priori* sur la fonction de fitness, on cherche à trouver le contrôleur qui maximise la valeur de performance, voire plusieurs valeurs si le problème est multi-objectif (e.g. [Pires *et al.* 2004, Molina-Cristóbal *et al.* 2005]), c'est-à-dire qui résout au mieux la tâche donnée dans l'environnement d'évaluation [Davidor 1991, Nolfi *et al.* 1994]. L'utilisation d'algorithmes évolutionnistes se justifie pleinement étant donné qu'il s'agit de méthodes d'optimisation stochastiques couramment appliquées à des problèmes d'optimisation boîte noire mono- ou multi-objectif [Droste *et al.* 2003, Pelikan *et al.* 2005, Droste *et al.* 2006].

2.3.1 Définition de la fonction de fitness et multi-objectivisation

Les algorithmes évolutionnistes cherchent les solutions qui maximisent le ou les objectifs de fitness. La définition de ces fonctions de fitness est donc une étape importante qui influe grandement sur le processus d'optimisation et sur la qualité des solutions trouvées et aussi sur leur originalité. Dans le cas idéal, une fonction de fitness doit juger de la capacité d'un comportement correspondant à un contrôleur donné à résoudre la tâche visée. Ainsi, tout comportement qui résoud la tâche sera récompensé et sélectionné sans a priori par le processus d'optimisation. En pratique, on peut distinguer plusieurs types de fonctions de fitness utilisés dans les expériences de robotique évolutionniste, notamment par la quantité d'information a priori sur le contrôleur optimal que le concepteur veut intégrer au processus. On peut distinguer 5 grands types de fonctions de fitness (pour une revue plus complète, se référer à [Nelson *et al.* 2009]) :

- Fitness obtenue sur des données d'apprentissage. En supposant qu'on dispose d'un certain nombre d'entrées pour lesquelles la réponse optimale du contrôleur est connue, l'erreur de prédiction entre la réponse effective du contrôleur et la réponse optimale attendue peut être utilisée comme fitness. Par exemple dans l'article [Mucientes *et al.* 2010], des comportements simples de navigation sont optimisés pour un robot à roues en minimisant l'erreur de prédiction sur les valeurs de moteurs en fonction des entrées sensorielles du robot sur plusieurs centaines de tests.
- Fitness comportementale. Une fitness comportementale indique pour un contrôleur donné si son comportement sur le robot vérifie un certain nombre de caractéristiques que doit posséder le comportement optimal. On se concentre donc ici uniquement sur le comportement du robot obtenu. Par exemple, des comportements de marche ont été obtenus par optimisation sur un robot octopode en définissant une fitness qui augmente ou diminue à chaque étape de l'évaluation en fonction de l'adéquation entre le comportement observé et des règles définissant le comportement optimal [Jakobi 1998b].
- Fitness par incréments fonctionnels. Ce type de fitness est typiquement utilisé pour faciliter l'apprentissage d'un comportement complexe en le découpant en différents sous-comportements. Le processus compte alors plusieurs étapes

d'optimisation successives où les contrôleurs sont optimisées pour apprendre un ensemble de ces sous-comportements, par exemple via une fitness comportementale. Cette techniquement a notamment été utilisée avec un robot portique pour l'apprentissage de suivi de cibles visuelles [Harvey *et al.* 1994] : le robot apprend d'abord à localiser des cibles fixes de grandes tailles, puis des cibles fixes de petites tailles et enfin des cibles mobiles à l'aide de 3 fonctions de fitness différentes.

- Fitness par incréments environnementaux. Cette fonction de fitness est basée sur les même principes qu'une fitness par incréments fonctionnels, à ceci près que la fonction de fitness n'est pas modifiée, alors que l'environnement d'évaluation se complexifie à chaque étape de l'optimisation : le robot doit alors résoudre la tâche considérée dans un environnement simple, puis un environnement un peu plus complexe, ... Bongard utilise une telle fonction de fitness dans un processus dit de *shaping* où un robot simulé apprend à se déplacer des objets de différentes formes et à les saisir [Bongard 2011a].
- Fitness dite agrégée. Une fitness agrégée contient un ou plusieurs termes ayant trait à la réalisation de la tâche. Par exemple si un robot doit ramasser un ensemble de balles dans une arène et les amener dans un panier, une fitness agrégée possible est le nombre de balles dans le panier à la fin de l'évaluation [Mouret & Doncieux 2011]. Ce type de fitness contient peu d'informations a priori sur la tâche à résoudre, étant donné qu'elle ne juge que du degré de réalisation de la tâche et non pas de la manière dont la tâche est réalisée.
- Absence de fitness. Dans certains travaux, aucune fonction de fitness n'est définie explicitement et on recherche les comportements les plus "nouveaux" sur la tâche considérée [Lehman & Stanley 2008, Lehman & Stanley 2010]. L'idée principale est de ne pas guider l'optimisation par un objectif de performance défini a priori par le concepteur, mais d'encourager l'exploration de l'espace comportemental afin de trouver des solutions de plus en plus originales, dont certains se révèlent également performantes. Cette recherche par nouveauté est décrite plus en détails dans la section 2.1.4.

La fonction de fitness définie peut parfois ne pas permettre de trouver des solutions adéquates. C'est notamment le cas quand les contrôleurs des premières générations obtiennent systématiquement des valeurs de fitness nulles : la fonction de fitness n'est pas exploitable en tant que pression de sélection. Ce problème dit de *bootstrap* peut par exemple intervenir quand la fonction de fitness est peu informative, comme dans le cas de fitness agrégées [Mouret & Doncieux 2008, Nelson *et al.* 2009, Mouret & Doncieux 2009b]. Il existe plusieurs méthodes pour résoudre ce problème de bootstrap, comme l'emploi de mesures de diversité (voir section 2.3.3). Dans certains expériences, plusieurs auteurs ont rapporté que l'optimisation de l'objectif de fitness peut être facilitée par l'introduction d'objectifs additionnels [Knowles *et al.* 2001, Jensen 2004, Handl *et al.* 2008, Mouret & Doncieux 2008, Mouret 2011] : au lieu d'optimiser seulement l'objectif de fitness principal, les solu-

tions sont sélectionnées en prenant également en compte d'autres objectifs [4]. Dans le cadre de cette méthode de multi-objectivisation, on peut :

- soit ajouter des objectifs auxiliaires qui ne sont pas directement liés à la résolution de la tâche (e.g. [Jensen 2004]), par exemple un objectif de diversité ou de nouveauté pour assurer une bonne exploration de l'espace de recherche (voir par exemple [Mouret 2011]) ;
- soit décomposer l'objectif de fitness principal en plusieurs sous-objectifs, ce qui peut limiter le nombre d'optima locaux et ainsi aider le processus d'optimisation (voir par exemple [Knowles *et al.* 2001, Handl *et al.* 2008, Mouret & Doncieux 2008]).

2.3.2 Vers une description comportementale

En robotique évolutionniste, la performance d'un contrôleur dépend de l'environnement d'évaluation [Harvey *et al.* 1993, Meyer *et al.* 1998] : en interagissant avec cet environnement d'évaluation, le phénotype donne lieu à un comportement spécifique dont la performance est notée par une valeur de fitness. Il est ainsi commode de définir, en plus du génotype, du phénotype et de la fitness, un niveau de description supplémentaire pour l'individu : le comportement (cf. figure 2.9). Contrairement aux autres niveaux de description d'un individu et comme illustré sur la figure 2.10, le passage du phénotype au comportement est une phase que l'utilisateur peut difficilement contrôler du fait de sa dépendance à l'environnement d'évaluation : la correspondance entre phénotype et comportement peut être complexe. Deux phénotypes très proches peuvent par exemple donner lieu à des comportements très différents et des phénotypes différents peuvent se comporter de manière identique.

Considérons un exemple simple où le génotype est un couple de bits (i, j) et le phénotype est la somme de ces 2 bits. Considérons les individus (1, 0) et (0, 1) : ces individus sont très différents d'un point de vue génotypique (2 bits inversés), mais donne lieu au même phénotype (1). Par contre, les individus (1, 0) et (1, 1) sont plus proches (1 bit inversé seulement), mais mènent à des phénotypes différents (respectivement 1 et 2). L'idée à retenir est qu'une distance définie sur l'espace des phénotypes peut ne pas présager de la proximité entre comportements.

Comprendre pour un problème donné la correspondance entre l'espace des génotypes/phénotypes et l'espace des comportements est un défi important en robotique évolutionniste [Mouret & Doncieux 2011] : introduire une description comportementale peut parfois rendre le processus d'optimisation plus efficace, par exemple pour maintenir la diversité des solutions [Trujillo *et al.* 2008, Gomez 2009, Doncieux & Mouret 2010, Ollion & Doncieux 2011].

4. Il ne s'agit pas à proprement parler d'un problème d'optimisation multi-objectif, étant donné que seule la performance obtenue sur l'objectif principal sera effectivement considérée à la fin du processus

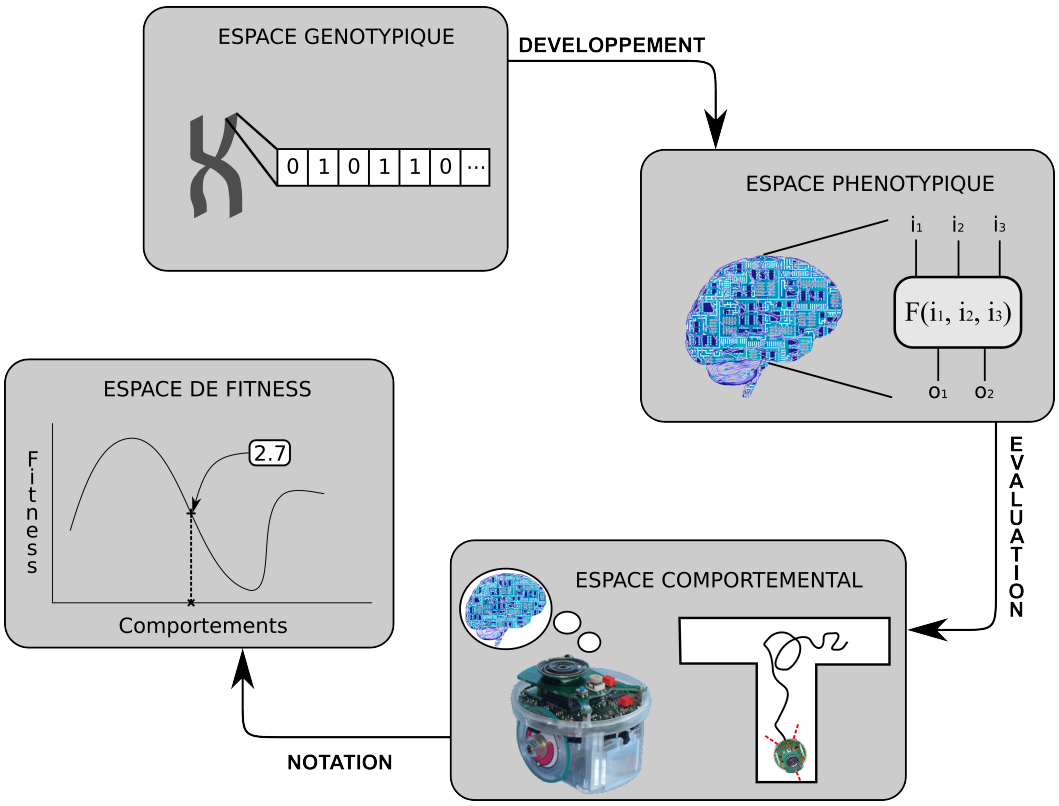

FIGURE 2.9 – Les différents niveaux de description d'une solution pour un algorithme évolutionniste.

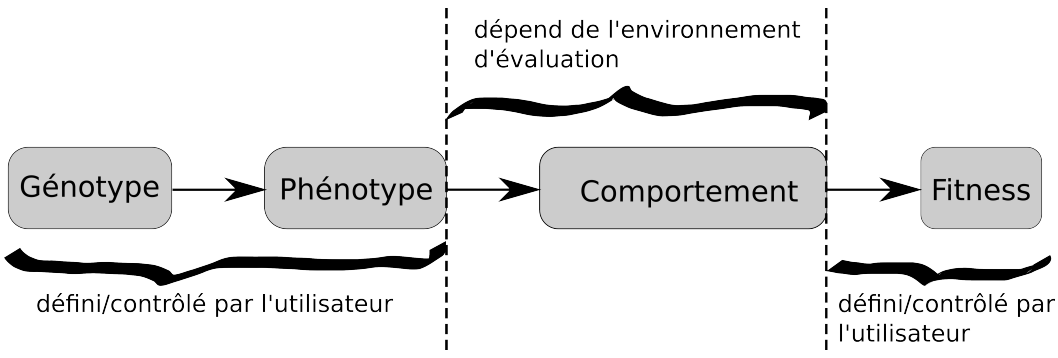

FIGURE 2.10 – En robotique évolutionniste, la correspondance entre phénotype et comportement est définie via l'environnement d'évaluation et peut être très complexe.

2.3.3 Diversité comportementale

Dans la littérature, la distance utilisée pour calculer le critère de diversité est souvent basée sur le génotype des solutions [Goldberg & Richardson 1987, Bäck & Hoffmeister 1991, Srinivas & Deb 1994, Stanley & Miikkulainen 2002, Ursem 2002, Toffolo & Benini 2003], par exemple en calculant une distance de Hamming entre deux chaînes de bits, ou parfois sur le phénotype [Deb & Goldberg 1989], par exemple avec une distance euclidienne entre deux vecteurs de valeurs réelles. Avec certains types de génotypes, il est néanmoins difficile de construire des distances performantes pour comparer les individus. Par exemple pour comparer des réseaux de neurones, il est possible de définir une distance entre graphes. On pourrait par exemple chercher le sous-graphe commun maximal entre deux graphes et ainsi évaluer le recouvrement des deux graphes [Bunke & Shearer 1998], mais il s'agit d'un problème NP-complet impliquant de recourir à des algorithmes de complexité élevée. D'autres travaux proposent de tenir le journal des mutations pour chaque individu pendant le processus évolutionniste et d'étudier la "parenté" entre individus [Stanley & Miikkulainen 2002]. Un tel critère de diversité proposé est rapide à calculer, mais il est difficile de comprendre de quelle façon ce critère va influencer le processus d'optimisation. D'autre part, bien que deux individus puissent être très différents d'un point de vue génotypique, leurs comportements peuvent être totalement identiques suivant la tâche à résoudre. En conséquence, pousser à la diversité génotypique peut potentiellement ne pas avoir d'influence sur la diversité effective de la population.

Récemment, plusieurs travaux ont étudié l'utilisation de distance comportementale pour comparer les solutions optimisées [Trujillo *et al.* 2008, Gomez 2009, Doncieux & Mouret 2010, Ollion & Doncieux 2011]. Dans ce cas, quels que soient leurs génotypes, deux individus de même comportement seront considérés comme identiques. De même, des individus aux génotypes proches mais réagissant de manières totalement différentes dans l'environnement d'évaluation pourront être plus facilement distingués. Dans diverses applications robotiques, il a été montré qu'utiliser un critère de diversité comportementale améliore significativement les résultats de l'optimisation en comparaison à l'utilisation d'une diversité génotypique [Gomez 2009, Mouret & Doncieux 2009b, Doncieux & Mouret 2010, Ollion & Doncieux 2011, Mouret & Doncieux 2011].

Pour calculer un critère de diversité, il faut souvent définir une fonction de distance qui permet de comparer les différents individus optimisés. Divers descripteurs comportementaux ont été proposés en robotique évolutionniste : trajectoire discrétisée [Trujillo *et al.* 2008, Doncieux & Mouret 2010, Ollion & Doncieux 2011], informations sensorimotrices [Doncieux & Mouret 2010], actions du robot au cours du temps [Gomez 2009], ... Des comparatifs entre plusieurs types de descripteurs indiquent que des distances relativement générales basées sur la trajectoire ou les informations sensorimotrices du robot permettent de définir des critères de diversité aussi performants que des distances plus *ad hoc* (voir par exemple [Gomez 2009, Mouret & Doncieux 2011]). Ceci suggère que le gain apporté

par la diversité comportementale n'est pas seulement lié à l'information que l'utilisateur injecte dans le processus d'optimisation en définissant la description comportementale utilisée.

2.4 Trois problèmes de robotique évolutionniste

Suite à cette introduction générale sur les algorithmes évolutionnistes multi-objectifs et leur utilisation pour développer des contrôleurs de robots dans un contexte d'optimisation boîte noire, nous allons nous concentrer dans la dernière partie de ce chapitre sur trois questions principales.

A. Pourquoi optimiser en simulation et comment optimiser un contrôleur en simulation qui se transfère bien sur le robot physique ?
B. Comment obtenir des contrôleurs de robot qui généralisent dans de nouvelles situations ?
C. Comment obtenir un robot qui s'adapte à des changements d'environnement ?

2.4.1 Des contrôleurs qui se transfèrent

Comme il est en autre difficile de déterminer ce que le robot perçoit de son environnement et donc quelles informations ou caractéristiques il aura plus de facilité à utiliser, un certain nombre de travaux proposent d'évaluer les contrôleurs directement sur le robot [Brooks 1991a, Brooks 1991b, Floreano & Mondada 1994]. En observant le comportement du robot, on pourra alors déterminer si la solution correspondante est plutôt intéressante ou non et calculer une note de performance. Ce processus présente également l'intérêt qu'un contrôleur peut exploiter directement des spécificités sensorielles et motrices du robot, même celles qui ne sont pas connues du concepteur. En pratique, effectuer l'ensemble des évaluations sur le robot pose certains inconvénients :

- Les évaluations s'effectuent nécessairement en temps réel et sont difficilement parallélisables sauf si d'autres robots "identiques" sont disponibles.
- Le comportement d'une solution n'est pas connu à l'avance : il peut se révéler dangereux, voire entraîner des dégâts sur le robot.
- L'état du robot n'est pas constant au cours du temps, ce qui implique que la performance d'une même solution peut changer pendant le processus d'optimisation.

Ces contraintes rendent souvent nécessaire le recours à une simulation pour évaluer les solutions. Les optima trouvés en simulation peuvent ne pas correspondre à des optima en réalité et on assiste alors à une perte de performance significative quand on passe de l'évaluation en simulation à l'évaluation sur le robot : c'est le problème du passage de la simulation à la réalité ou *reality gap* [Jakobi *et al.* 1995, Jakobi 1998a]. Ce problème est l'une des principales limitations de la robotique évolutionniste depuis son apparition [Brooks 1992, Harvey *et al.* 1993, Pollack *et al.* 2000].

Pour donner une vue claire des différents travaux traitant ce problème, nous allons découper cette partie en trois sous-sections regroupant respectivement :

2.4.1.1 les approches où l'optimisation se fait totalement sur le robot ;
2.4.1.2 les approches où l'optimisation se fait essentiellement en simulation ;
2.4.1.3 les approches hybrides où l'optimisation se fait en simulation, mais prend également en compte quelques informations venant de la réalité.

2.4.1.1 Optimiser directement sur le robot

Comme évaluer une solution sur un robot peut se révéler long et fastidieux, l'utilisation d'un algorithme évolutionniste (optimisant donc une population de contrôleurs) oblige souvent d'utiliser une population de solutions restreinte et de limiter également le nombre de générations. On peut citer, par exemple, le travail de Gongora et al. en 2009 qui ont appliqué un algorithme évolutionniste à l'optimisation de contrôleurs pour un petit hélicoptère avec une population de 20 individus optimisés pendant 30 générations, soit 600 évaluations sur le robot seulement [Gongora *et al.* 2009]. Le dispositif expérimental nécessitait de plus d'attendre plusieurs minutes entre les générations pour éviter les problèmes de surchauffe de l'appareil. Des expériences similaires ont été effectuées sur un robot mobile Khepera naviguant dans un labyrinthe dans le but de trouver des contrôleurs robustes à une série de perturbations de l'environnement [Floreano & Mondada 1998]. L'optimisation en elle-même prenait une soixantaine d'heures pour un total de 8000 évaluations sur le robot. De même, des contrôleurs de marche ont été optimisés pour des robots Sony AIBO [Hornby *et al.* 2002, Kohl & Stone 2004] (1000 évaluations en 3 heures pour [Kohl & Stone 2004]), sur un robot à 9 pattes [Zykov *et al.* 2004] (900 évaluations sur près d'1 heure 30), sur un robot quadrupède [Yosinski *et al.* 2011] (entre 500 et 1200 évaluations suivant les expériences) ou encore sur un robot humanoïde [Hebbel *et al.* 2006] (environ 6000 évaluations en plusieurs heures). On peut également citer un travail d'optimisation de la charge utile pouvant être portée par un robot ornithoptère [Regan *et al.* 2006] avec 3000 évaluations effectuées sur le robot. En comparaison, des processus d'optimisation multi-objectif via un algorithme évolutionniste peuvent nécessiter plusieurs dizaines de milliers d'évaluations (e.g. [Zitzler *et al.* 2000, Deb *et al.* 2002b]) et certaines expériences de neuro-évolution autorisent même jusqu'à plusieurs millions d'évaluations (e.g. [Ruppin 2002]).

En évaluant les solutions sur le robot, on peut penser que le gradient obtenu pour l'optimisation est exact. Or, le robot étant une entité physique, il est susceptible de varier au cours du temps : les moteurs peuvent s'user et surchauffer, les pièces peuvent se déformer, voire casser, ... Ces différentes sources de variabilité peuvent sensiblement bruiter la valeur de performance attribuée à chaque solution. Par exemple, une solution excellente testée à un moment où le robot fonctionne mal sera rejetée au détriment d'une solution médiocre évaluée au bon moment. Dans certains cas, l'optimisation peut avoir lieu sur le robot mais dans un environnement d'évaluation simplifié (e.g. [Gongora *et al.* 2009]) : des pertes de performance

peuvent alors être observés en passant de l'environnement simplifié à l'environnement complet, bien que l'évaluation des solutions ait lieu directement sur le robot.

2.4.1.2 Optimiser en simulation

Afin de réduire le nombre d'évaluations sur le robot, il est possible d'effectuer une partie de l'optimisation en simulation avant de terminer le processus sur le robot [Lipson & Pollack 2000, Pollack & Lipson 2000]. Par exemple, dans le cadre du projet GOLEM visant à co-évoluer morphologies et contrôleurs, le processus d'optimisation comportait deux phases. Durant la première, morphologies et contrôleurs étaient évoluées dans une simulation. Dans un second temps, la meilleure morphologie était sélectionnée et l'optimisation de contrôleurs s'effectuaient sur une population de robots physiques construits suivant cette morphologie. Des travaux reposant sur le même type de techniques ont été menés sur des tâches de navigation avec un robot mobile Khepera [Nolfi *et al.* 1994, Miglino *et al.* 1995] : un contrôleur de navigation est optimisé pendant 30000 évaluations en simulation, puis pendant 3000 évaluations sur le Khepera [Nolfi *et al.* 1994] (20000 évaluation en simulation et 2000 en réalité pour [Miglino *et al.* 1995]).

Ces travaux reposent sur l'hypothèse implicite que l'optimisation en simulation va guider les solutions vers des zones relativement proches des optima en réalité. D'après les résultats présentés dans les articles [Nolfi *et al.* 1994] et [Miglino *et al.* 1995], les meilleures solutions trouvées en simulation obtiennent en effet une fitness raisonnable sur le robot, malgré une perte de performance : en exploitant la fonction de fitness en simulation, l'optimisation permet de trouver des contrôleurs non triviaux proches des contrôleurs optimaux en réalité. Ces méthodes ne vont probablement pas bien fonctionner dans le cas où la solution trouvée en simulation n'est pas du tout performante une fois transférée sur le robot.

Un simulateur est souvent un compromis entre le temps d'évaluation des solutions et sa précision. En effet, un simulateur aura beau simuler parfaitement le système réel, il n'aura pas d'utilité si le temps d'évaluation d'une solution est trop long. *A contrario*, accélérer l'évaluation des solutions se fera souvent au détriment de la précision du simulateur : certains phénomènes physiques vont être approchés, voire complètement ignorés. Ce problème a notamment été soulevé par Cliff et al. dans le cas de simulateurs de systèmes de vision pour des robots [Cliff *et al.* 1993a]. Dans certains cas, il est même impossible d'avoir un simulateur "parfait" : par exemple, la locomotion de certains robots mobiles peut impliquer des interactions roues-sol ou pattes-sol complexes et les modèles de simulation utilisés sont souvent basés sur des hypothèses simplificatrices [Weingarten *et al.* 2004, Lhomme-Desages *et al.* 2007, Terekhov *et al.* 2010a].

Afin d'obtenir un processus d'optimisation rapide, certains chercheurs ont proposé d'évaluer les contrôleurs uniquement via une simulation en prenant en compte les problèmes de transfert de la simulation à la réalité [Jakobi *et al.* 1995, Jakobi 1997, Saunders *et al.* 2011]. Ainsi, partant du principe que le simulateur utilisé ne va pas toujours être précis, plusieurs méthodes ont été développées pour

s'assurer que les solutions optimisées en simulation soient également performantes sur le robot, c'est-à-dire que la solution finale soit robuste au passage de la simulation à la réalité. La plus formalisée des ces approches est le recours à une simulation minimale proposé par Jakobi [Jakobi 1997]. Cette technique consiste à ne modéliser que les aspects de la réalité qui vont avoir un sens pour la tâche à résoudre. Les phénomènes physiques liés à des comportements peu performants ou instables vont être soit ignorés, soit suffisamment bruités pour que les solutions ne puissent pas les exploiter. Un exemple typique est celui de la marche d'un octopode [Jakobi 1998b]. La collision entre pattes qui est un phénomène complexe à modéliser est simplement prise en compte par une pénalité sur la fitness, étant donné que les comportements mettant en jeu une collision ne seront pas intéressants *a priori*. Ainsi, aucun comportement ne pourra exploiter en simulation la collision des pattes pour obtenir des performances élevées irréalistes et les comportements sans collision seront sélectionnés.

Pénaliser des comportements quand ils exploitent certains phénomènes n'est possible que si l'utilisateur a déjà une idée de quel sera le comportement optimal : il va alors chercher à biaiser l'optimisation vers ce type de comportements. Néanmoins, un des intérêts des algorithmes évolutionnistes en tant que méthodes d'optimisation réside justement dans le fait qu'ils nécessitent peu d'informations *a priori* sur le problème à résoudre à part la fonction de fitness et peuvent donc résoudre le problème de manière "créative". En contraignant la recherche vers un comportement optimal précis, on perd cet aspect créatif.

Les autres techniques utilisées en robotique évolutionniste pour obtenir des solutions robustes par rapport aux spécificités de l'environnement simulé reposent essentiellement sur l'utilisation de bruit. Par exemple, la robustesse peut être obtenue en évaluant chaque solution dans plusieurs simulateurs [Jakobi 1997, Jakobi 1998a, Thompson *et al.* 1999]. Afin de trouver des comportements relativement peu dépendants des caractéristiques spécifiques de la simulation, Jakobi propose notamment d'évaluer les solutions dans 10 simulateurs, chacun modélisant la tâche à résoudre avec des valeurs de paramètres différentes. Cette méthode fut par exemple appliquée à une tâche de navigation dans un labyrinthe en T avec un robot mobile Khepera [Jakobi 1997, Jakobi 1998a] : en faisant varier la taille du labyrinthe, la position initiale du robot et d'autres facteurs en simulation, l'optimisation finit toujours par trouver un comportement robuste qui se transfère correctement sur le robot physique. Des expériences similaires ont été réalisées avec un robot portique devant réagir en fonction de stimuli perçus par une caméra [Jakobi 1997, Jakobi 1998a]. On peut remarquer que le problème de passage de la simulation à la réalité est ici principalement lié aux capteurs du robot : étant donné que les valeurs des capteurs sont bruitées, un contrôleur basé sur des valeurs précises de ces capteurs aura de grandes chances de mal se transférer sur le robot physique. Dans l'expérience du robot Khepera naviguant dans un labyrinthe en T, le comportement optimal trouvé est celui où le robot utilise ses capteurs le moins : le robot va tout droit dans le premier corridor, détecte le mur du fond et tourne. Une méthode plus directe revient à modéliser les capteurs en prenant en compte le niveau de bruit observé en réalité.

Il a d'ailleurs été montré, également sur un robot mobile Khepera, que des niveaux de bruit réalistes permettaient dans certains cas un passage de la simulation à la réalité sans heurt [Jakobi *et al.* 1995, Miglino *et al.* 1995]. Les travaux de Funes et Pollack sur l'évolution de formes construites en pièces de Lego mettent en jeu un mécanisme similaire [Funes & Pollack 1999] : afin de déterminer si une configuration optimisée va bien se transférer en réalité, la solidité des différentes pièces est sous-estimée de 20 % en simulation pour prendre en compte les incertitudes sur la fiabilité du simulateur. Bien que ce pourcentage soit arbitraire, il permet de trouver des formes originales constructibles en réalité.

Ces méthodes peuvent néanmoins difficilement s'appliquer au cas où le passage de la simulation à la réalité provient de la dynamique du robot. Si la dynamique du robot n'est pas correctement modélisée dans certaines parties de l'espace des comportements, il est difficile de trouver un ensemble de paramètres à faire varier pour obtenir des comportements robustes. Plus généralement, il semble difficile de pouvoir définir, pour chaque tâche, un ensemble de paramètres permettant de trouver des comportements suffisamment robustes en simulation. Là encore, l'utilisateur doit mettre en évidence les éléments de la simulation qui vont poser problème et différer significativement de la réalité (la solidité des différentes pièces par exemple) avant de pouvoir lancer l'optimisation. D'autre part, évoluer les solutions dans plusieurs simulations mène à un coût en temps de calcul qui est directement lié au nombre de paramètres à varier.

D'autres approches intègrent directement la robustesse dans la structure du contrôleur optimisé : le robot va toujours chercher à s'adapter à son environnement, qu'il soit en simulation ou en réalité. Ainsi, le passage de la simulation à la réalité est perçu par le robot comme un changement d'environnement auquel il doit s'adapter. On peut par exemple utiliser des réseaux de neurones plastiques [Urzelai & Floreano 2001, Floreano & Urzelai 2001] : au début de chaque évaluation, les poids du réseaux sont fixés aléatoirement et des règles d'adaptation entrent en jeu pour faire varier ces poids au cours du temps d'évaluation. Le génotype n'encode alors plus les poids du réseaux, mais les règles d'adaptation qui vont s'appliquer sur les différentes connexions. Cette technique a été appliquée à une tâche simple où un robot doit naviguer dans une arène pour allumer une lampe avant de s'en approcher : bien que la perte de performance lors du passage de la simulation à la réalité soit toujours importante, le robot physique obtient un comportement plus efficace qu'avec un réseau classique de neurones à poids fixes. Une autre approche revient à construire explicitement un modèle de l'environnement simulé et de corriger le comportement du contrôleur lorsque des différences sont perçues entre l'environnement actuel du robot et le modèle. Par exemple, Hartland et Bredèche proposent d'intégrer deux modules au contrôleur [Hartland & Bredèche 2006] :

- un module d'anticipation qui prédit l'état suivant du robot en fonction de son état actuel et des valeurs envoyées aux moteurs ;
- un module de correction ajouté quand le contrôleur est transféré sur le robot et qui fait s'adapter le comportement du robot en fonction des différences perçues entre l'état des moteurs prédit et l'état des moteurs observé.

Cette méthode a été appliquée à un robot Khepera doté d'une caméra devant se déplacer vers un repère visuel de l'environnement. Que la méthode utilise de la plasticité ou cherche à anticiper les états du robot, le passage de la simulation vers la réalité est remplacé par une étape d'adaptation. Autrement dit, si le contrôleur transféré obtient des performances très différentes entre la simulation et la réalité, le mécanisme d'adaptation va partir du comportement peu performant en réalité et tenter de l'améliorer. En conséquence, il n'y a pas de garantie que le comportement final trouvé ait grand rapport avec le comportement initial simulé et il est vraisemblable que ce type de méthodes ne permettent que de converger vers un optimum local en réalité situé à proximité de la solution transférée.

2.4.1.3 Optimiser en simulation à l'aide du robot

Le troisième type d'approche que nous allons à présent détailler ne doit pas être confondu avec celles visant à optimiser les contrôleurs directement sur le robot. Dans le cas qui nous intéresse maintenant, l'optimisation se fait uniquement en simulation, mais quelques expériences menées sur le robot physique permettent également de guider la recherche. Cette partie regroupe les approches qui cherchent à générer de l'information sur le robot afin de trouver des solutions performantes qui se transfèrent bien. On peut distinguer deux classes de méthodes :

- les méthodes visant à améliorer un simulateur du robot de manière automatique avant ou pendant la recherche d'un contrôleur intéressant [Bongard & Lipson 2004, Bongard & Lipson 2005, Zagal & Ruiz-Del-Solar 2007, Zagal *et al.* 2008, Zagal *et al.* 2009] ;
- les méthodes basées sur un modèle de substitution [Kumano *et al.* 2006, Abbeel *et al.* 2007, Hemker *et al.* 2009, Abbeel *et al.* 2009], où une approximation de la fonction de fitness est directement construite à l'aide de quelques expériences en réalité.

Afin de trouver des contrôleurs performants et qui se transfèrent bien sur le robot, une possibilité est d'améliorer le simulateur du robot jusqu'à ce qu'il soit suffisamment précis pour éviter les problèmes de passage à la réalité. Certains travaux proposent par exemple d'améliorer le simulateur via des méthodes de coévolution [Bongard & Lipson 2005, Zagal *et al.* 2008] utilisant deux populations : une de modèles de simulation et une autre d'expériences-tests. D'un côté, les modèles sont optimisés en fonction des données des expériences-tests déjà effectuées sur le robot et de l'autre, on cherche les expériences-tests les plus informatives vis-à-vis des meilleurs modèles construits. Par exemple, dans les articles [Bongard & Lipson 2004, Bongard & Lipson 2005], un algorithme dit d'estimation-exploration est utilisé pour co-évoluer une population de modèles de morphologie d'un robot quadrupède avec une population de réseaux de neurones utilisés pour contrôler le robot. Dans ce cas, une expérience-test est le transfert d'un contrôleur donné sur le robot. L'algorithme alterne deux phases d'optimisation : (1) la phase d'estimation où les modèles de simulation sont optimisés pour prédire au mieux les séries temporelles des capteurs du robot observées pendant les expériences-

tests déjà effectuées sur le robot ; (2) la phase d'exploration où on cherche le test qui maximise un critère donné. Pour choisir la prochaine expérience-test à effectuer, plusieurs critères peuvent être définies. On peut par exemple chercher le contrôleur ayant les meilleures performances [Bongard & Lipson 2004] : on va chercher à améliorer le modèle de simulation là où les comportements sont les plus intéressants. Une alternative est de choisir le test qui va maximiser le désaccord entre les prédictions des meilleurs modèles construits jusque là [Bongard & Lipson 2005] : on cherche alors à construire un modèle plus global. Un autre travail inspiré par cet algorithme a été appliqué au contrôle de l'attitude d'un quadri-rotor simulé [Koos *et al.* 2009] : les contrôleurs-tests sont cette fois évalués par leur performance et un indice de désaccord dans le cadre d'un algorithme multi-objectifs. Ceci permet de chercher des tests assurant un compromis entre l'apport d'information aux modèles et leur performance sur la tâche à résoudre et donc de biaiser l'optimisation des modèles vers les zones d'intérêt de l'espace de recherche.

Basé également sur un processus de coévolution entre simulateurs et contrôleurs, l'algorithme "Back-to-Reality" cherche un modèle qui va prédire au mieux la fitness en réalité [Zagal & Ruiz-Del-Solar 2007, Zagal *et al.* 2008]. Les contrôleurs sont évalués grâce au meilleur modèle construit jusque là et, une fois que la population a convergé, ils sont tous transférés sur le robot. Les variations de fitness des meilleurs individus en réalité sont alors utilisées pour optimiser un meilleur modèle, l'idée étant de prédire de mieux en mieux la fitness tout en optimisant la performance des contrôleurs. Cette approche a été appliquée à une tâche de frappe de balle avec un robot Sony AIBO [Zagal *et al.* 2008], ainsi qu'à de la marche humanoïde avec un robot Nao [Zagal *et al.* 2009].

D'un point de vue général, ces méthodes visant à améliorer le simulateur reposent toutes sur l'hypothèse que le modèle de simulation peut devenir suffisamment bon pour permettre des transferts performants du simulateur vers le robot. En pratique, cette hypothèse est vérifiée si on modélise des dynamiques simples ou en ajustant un certain nombre de paramètres sur une structure de modèle fixe, mais ce n'est vraisemblablement pas le cas pour des robots aux dynamiques complexes et où l'identification du modèle sera difficilement envisageable de manière automatique. Par exemple, imaginons que nous utilisons un simulateur dynamique basé sur *Open Dynamics Engine* (ODE, [Smith 2005]) pour modéliser le comportement d'un robot volant. Comme ODE ne rend pas compte de la mécanique des fluides, rendre un tel modèle de simulation fiable reviendrait à redécouvrir les lois de l'aérodynamique. L'identification du modèle de simulation peut donc être longue et coûteuse en temps expérimental, ce qui n'est pas envisageable si on cherche un processus d'optimisation rapide.

En-dehors des méthodes visant à améliorer un simulateur, l'optimisation peut s'effectuer sur un modèle de substitution ou *surrogate model*[5] qui approxime directement la fonction de fitness réelle [Jones 2001, Jin 2005, Kumano *et al.* 2006,

5. La notion de modèles de substitution sera abordée plus en détail dans le chapitre 3, notamment dans un contexte d'optimisation.

Hemker *et al.* 2009]. Ce modèle peut être mis à jour pendant l'optimisation à l'aide d'une stratégie de mise à jour qui va indiquer quelles expériences doivent être effectuées pour améliorer au mieux le modèle d'après un critère donné [Jones 2001, Jin 2005]. Le but est alors de construire une fonction qui permet de relier directement l'espace des génotypes à l'espace de la fitness réelle en effectuant un minimum d'évaluations sur le robot [Kumano *et al.* 2006, Abbeel *et al.* 2007, Hemker *et al.* 2009]. Ainsi, au lieu de modéliser l'environnement et le robot à l'aide d'une simulation dynamique, on va juste chercher à approximer la relation génotype-fitness. Bien que les modèles de substitution soient souvent utilisés dans des travaux impliquant des algorithmes évolutionnistes, leur utilisation en robotique évolutionniste n'est pas encore très répandue. Par exemple, dans l'article [Kumano *et al.* 2006], des modèles de substitution construit par des méthodes de *kriging* (voir partie 3.1.5) sont optimisés via un algorithme évolutionniste multi-objectifs afin de concevoir des formes d'ailes performantes pour de petits avions. En-dehors de la robotique évolutionniste, on peut notamment citer les travaux de Hemker et al. sur la marche rapide d'un robot humanoïde HR18 [Hemker *et al.* 2006, Hemker *et al.* 2009]. Chaque contrôleur est défini par 5 paramètres réels définissant le comportement de marche du robot et le modèle de substitution est construit à l'aide d'une méthode d'interpolation par *kriging*. L'optimisation permet de trouver une marche où le robot se déplace à 40 cm/s après une centaine d'expériences effectuées en quelques heures. Des méthodes similaires ont été testées pour l'apprentissage de comportements de voltige avec un hélicoptère télécommandé à l'aide de techniques d'apprentissage par renforcement [Abbeel *et al.* 2007]. Un modèle approché de la dynamique du robot est initialement construit à l'aide de trajectoires générées par un pilote. Ce modèle est ensuite utilisé pour trouver une politique de contrôle optimale qui est alors testée sur le robot. Les nouvelles trajectoires générées permettent de mettre à jour le modèle local et le processus est itéré jusqu'à ce que la politique trouvée fonctionne suffisamment bien sur l'hélicoptère. Cette technique a aussi été adaptée à l'apprentissage de comportements d'urgence en cas de défaillance d'un moteur [Abbeel *et al.* 2009]. Les résultats obtenus sont plutôt impressionnants. Cette méthode n'est cependant applicable que si des données non triviales peuvent être générées à l'aide du robot pour initialiser le modèle. Suivant le robot étudié, cette étape peut nécessiter un nombre important d'expériences sur le robot.

2.4.1.4 Conclusions

De ces différents travaux, on peut dériver 5 idées principales reliées au problème du passage de la simulation à la réalité.

- L'optimisation menée directement sur le robot, bien que séduisante, est synonyme de processus d'optimisation lents et coûteux. Des comportements dangereux peuvent parfois être transférés sur le robot.
- Compléter une optimisation en simulation par quelques générations sur le robot physique n'est envisageable que si les solutions optimales en simulation sont relativement proches des solutions optimales en réalité.

- Il n'y a pas de garantie qu'un contrôleur robuste en simulation le soit suffisamment pour bien se transférer sur le robot.
- Des mécanismes d'adaptation directement intégrés dans le contrôleur risquent de ne pas retrouver de comportements efficaces en réalité après transfert si la perte de performance au moment du transfert est très importante.
- Les approches cherchant à optimiser les contrôleurs en simulation à l'aide du robot physique nous semble actuellement les approches les plus prometteuses. Néanmoins, il est généralement difficile d'optimiser suffisamment un simulateur ou un modèle de substitution pour permettre un bon transfert, notamment si le modèle est généré de zéro.

2.4.2 Des contrôleurs qui généralisent

La deuxième question que nous allons considérer dans ce manuscrit concerne la capacité de généralisation des contrôleurs. Supposons par exemple qu'un robot ait à effectuer une tâche ménagère dans une pièce : sa performance ne doit pas dépendre de la taille de pièce, de sa position initiale, de son orientation initiale, ... Or, en robotique évolutionniste, les contrôleurs sont très souvent évalués dans moins de 10 contextes [6] (e.g. 3 dans [Ziemke & Thieme 2002, Mouret & Doncieux 2011], 10 dans [Jakobi 1998a]). En conséquence, même si de bonnes performances sont observés dans le contexte spécifique d'évaluation, il n'y a aucune garantie que des performances similaires soient observés en modifiant certains aspects du contexte. Dans de nombreuses applications robotiques, le robot doit pourtant gérer de nombreuses situations différentes des situations envisagées lors de l'optimisation de son contrôleur : ce contrôleur doit donc être doté de capacités de généralisation suffisantes afin de pouvoir générer un comportement correct dans toutes les situations. Le faire de pouvoir doter ou non les contrôleurs générés par évolution de capacités de généralisation conditionne l'applicabilité des techniques de robotique évolutionniste à des applications robotiques réalistes [Harvey *et al.* 1993, Jakobi 1997, Jakobi 1998a].

2.4.2.1 Formaliser la capacité de généralisation

Avant d'introduire les différents travaux effectués sur l'amélioration des capacités de généralisation des contrôleurs, nous allons d'abord proposer une définition formelle de ce que nous entendons par capacité de généralisation. Comme le phénomène de généralisation a beaucoup plus été étudié dans le cadre de l'apprentissage supervisé, notre définition s'inspire de la méthodologie standard en apprentissage supervisé définissant trois ensembles de données [Alpaydin 2004, Gagné & Schoenauer 2006] :

- un ensemble de contextes d'apprentissage Ω_{train} utilisé pour optimiser les solutions ;

6. Un contexte peut souvent être décrit comme un vecteur de valeurs spécifiques attribuées à chaque paramètre de la tâche et nous nous limiterons à cette définition dans cette section.

- un ensemble de contextes de validation Ω_{valid} permettant d'estimer la capacité de généralisation de solutions prometteuses sur Ω_{train} ;
- un ensemble de contextes de test Ω_{test} permettant, une fois l'optimisation terminée, d'évaluer la capacité de généralisation sur de nouveaux contextes des meilleures solutions trouvées.

Dans le cadre de la robotique évolutionniste, un contexte ω est une instance de la tâche à résoudre et correspond à un ensemble de valeurs spécifiques pour chacun des paramètres de la tâche : par exemple, les coordonnées initiales du robot, les dimensions de l'environnement, ... Comme à la fois l'ensemble d'apprentissage Ω_{train} et l'ensemble de validation Ω_{valid} sont utilisés pour évaluer les solutions pendant l'optimisation, nous introduisons aussi la notion d'ensemble d'évaluation $\Omega_{eval} = \Omega_{train} \cup \Omega_{valid}$ qui contient tous les contextes pouvant servir à évaluer une solution pendant l'optimisation.

Soit $\mathcal{F}$ la fonction de fitness associée à la tâche et x un individu quelconque de l'espace des contrôleurs $\mathcal{C}$, la capacité de généralisation G de x sur un ensemble de contextes Ω est simplement définie comme la somme des performances de x sur toutes les contextes de Ω :

$$G(x, \Omega) = \sum_{\omega \in \Omega} \mathcal{F}(x, \omega)$$

Un individu optimalement général x doit alors vérifier les deux contraintes ci-dessous :

$$\forall y \in \mathcal{C}, \quad G(y, \Omega_{eval}) \leq G(x, \Omega_{eval}) \tag{C1}$$

$$\frac{G(x, \Omega_{eval})}{|\Omega_{eval}|} \simeq \frac{G(x, \Omega_{test})}{|\Omega_{test}|} \tag{C2}$$

Une solution optimalement générale doit donc se comporter au mieux sur les contextes d'évaluation issus de Ω_{train} et Ω_{valid} (C1) et obtenir des performances similaires sur les contextes de test (C2).

2.4.2.2 Promouvoir la capacité de généralisation sans l'évaluer

Certaines approches de robotique évolutionniste visent à trouver des contrôleurs robustes aux spécificités de l'environnement d'évaluation, tout en n'évaluant les solutions que dans un unique contexte. Parmi celles-ci, l'approche proposée par Reynolds [Reynolds 1994] consiste à ajouter du bruit sur les entrées des contrôleurs. Dans ce travail, le but est d'obtenir des comportements de suivi de couloirs robustes à la variabilité des valeurs indiquées par les capteurs de distance d'un robot simulé. Des approches similaires ont été appliquées au problème du passage de la simulation à la réalité [Miglino *et al.* 1995, Jakobi *et al.* 1995] : afin de trouver des contrôleurs efficaces sur le robot physique (qui fait office de contexte de test), du bruit est utilisé pour gommer les spécificités de la simulation. Les solutions optimisées ne peut alors pas exploiter ces spécificités pour obtenir de bonnes performances et développent donc des comportements plus robustes.

Une alternative proposée par Floreano et al. consiste à intégrer des mécanismes d'adaptation au niveau du contrôleur, par exemple en utilisant des réseaux de neurones dits plastiques [Floreano & Mondada 1998, Floreano & Urzelai 2001] où à chaque neurone est associée une règle d'apprentissage hebbienne modifiant les différents poids synaptiques du neurone au cours de l'évaluation. De tels réseaux permettent notamment de s'adapter rapidement à des configurations aléatoires de l'environnement dans une tâche de navigation avec un robot mobile Khepera [Urzelai & Floreano 2001]. A chaque génération, la performance d'un réseau est calculée dans un seul contexte.

En évaluant les contrôleurs dans un unique contexte, le processus d'optimisation ne peut exploiter aucune information sur la capacité de généralisation des solutions. Il n'y a donc aucune garantie qu'au terme de l'optimisation, les contrôleurs obtenus soient effectivement robustes.

2.4.2.3 Promouvoir les capacités de généralisation en les évaluant

Un échantillonnage d'articles en robotique évolutionniste (cf. table 2.1) met en évidence que, bien que de nombreux travaux se soient intéressés à comment optimiser des contrôleurs robustes, relativement peu évaluent la capacité de généralisation des contrôleurs pendant l'optimisation. Et dans le cas où la capacité de généralisation est effectivement évaluée, les contrôleurs sont souvent évalués dans un petit nombre de contextes d'évaluation (moins de 10, cf. table 2.1). Par exemple, dans l'article [Mouret & Doncieux 2011], des contrôleurs sont optimisés pour une tâche de collecte de balles dans une arène en évaluant chaque individu avec 3 positions initiales fixes. De même dans l'article [Ziemke & Thieme 2002], des contrôleurs ont été optimisés pour une tâche de navigation dans 6 contextes d'évaluation avec différentes formes de labyrinthe et différentes positions initiales du robot. Dans ces travaux, on cherche des contrôleurs performants sur l'ensemble des contextes d'évaluation, mais les capacités de généralisation des solutions obtenues ne sont pas estimées dans de nouveaux contextes après l'optimisation.

Certains travaux combinent à la fois l'intégration de mécanismes d'adaptation au niveau du contrôleur ainsi que l'utilisation de plusieurs de contextes d'évaluation. Par exemple, Kondo et al. présentent une approche basée sur des réseaux de neurones à réarrangement dynamique [Kondo *et al.* 1999] utilisés pour contrôler un robot mobile Khepera et dont les poids synaptiques peuvent varier en fonction de l'environnement. Les contrôleurs sont optimisés dans 10 contextes d'évaluation avec différentes orientations initiales du robot. Les meilleures solutions obtiennent de bonnes performances quand ils sont testés dans de nouveaux contextes où l'environnement et la dynamique du robot sont modifiés.

D'autres méthodes consistent à utiliser un ensemble de contextes d'évaluations variable pendant l'optimisation. On peut citer notamment le travail de Jakobi [Jakobi 1997, Jakobi 1998a] dont le but initial est de trouver des contrôleurs robustes afin qu'ils se transfèrent efficacement d'un simulateur à un robot physique et soient également robustes sur le robot. Par exemple, dans une tâche de

TABLE 2.1 – Echantillonnage d'articles traitent de problèmes de robotique évolutionniste. Les colonnes correspondent aux trois questions suivantes. (1) La capacité de généralisation des contrôleurs est-elle évaluée au cours de l'optimisation ? (2) Si oui, dans combien de contextes d'évaluation ($|\Omega_{eval}|$) ? (3) La capacité de généralisation des solutions est-elle testée après l'optimisation (Ω_{test}).

Papiers	Eval. de la généralisation ?		$\lvert\Omega_{eval}\rvert$	Ω_{test}
	non	oui		
[Jakobi *et al.* 1995]	x		1	x
[Miglino *et al.* 1995]	x		1	
[Jakobi 1997]		x	10	x
[Kondo *et al.* 1999]		x	10	x
[Di Paolo 2000]	x		1	x
[Floreano & Urzelai 2001]	x		1	x
[Berlanga *et al.* 2002]		x	6	x
[Ziemke & Thieme 2002]		x	3	
[Barate & Manzanera 2008]		x	4	x
[Mouret & Doncieux 2011]		x	3	
[Lehman & Stanley 2010]	x		1	
[Bongard 2011b]	x		1	x

navigation d'un robot mobile Khepera dans un labyrinthe en T [Jakobi 1997], les contrôleurs sont évalués indépendamment dans 10 simulateurs aux paramètres variables : chaque valeur de paramètre est générée aléatoirement dans un ensemble de valeurs donné. Une alternative est de générer l'ensemble d'évaluation par coévolution [Berlanga *et al.* 2002], où l'ensemble d'évaluation Ω_{eval} est un sous-ensemble généré à partir d'une liste de tous les contextes possibles. Le meilleur individu trouvé pendant l'optimisation est ensuite testé sur l'ensemble des contextes possibles. Bien qu'il ait une estimation de la capacité de généralisation, on remarquera que l'ensemble de test utilisé est identique à l'ensemble d'évaluation : les capacités de généralisation des solutions sont testées sur des contextes potentiellement utilisés pendant l'évaluation. L'estimation est donc biaisée, alors que les ensembles Ω_{eval} et Ω_{test} devraient être indépendants pour avoir une meilleure idée des capacités de généralisation des contrôleurs obtenus.

Le travail de Barate et Manzanera se rapproche plus de la formalisation classique de l'apprentissage supervisé [Barate & Manzanera 2008]. Le but est d'optimiser des contrôleurs évitant des obstacles dans une arène à l'aide de capteurs visuels. Les individus sont évalués sur 4 contextes où la position initiale du robot varie (Ω_{eval}). Les solutions trouvées à la dernière génération sont ensuite évaluées sur 4 nouveaux contextes avec des configurations différentes d'obstacles (Ω_{test}) et la meilleure solution est celle qui obtient les meilleures performances sur ces nouveaux contextes, autrement dit celle qui généralise le mieux. Comme les ensembles d'évaluation et de test sont disjoints, ce protocole est plus adapté pour effectivement estimer la capacité de généralisation des contrôleurs obtenus. Le désavantage principal d'une telle méthode est que tous les individus sont évalués sur Ω_{eval}, ce qui limite la taille de l'ensemble d'évaluations utilisable sous peine de ralentir drastiquement l'optimisation.

2.4.2.4 Réduire le nombre d'évaluations

Afin de vérifier la contrainte C1, une méthode directe revient à optimiser les solutions directement sur Ω_{eval}. Néanmoins, ceci contraint la taille de l'ensemble d'évaluations, car chaque solution doit être évaluée sur chaque contexte de Ω_{eval}. Ainsi, si l'utilisateur souhaite définir un ensemble d'évaluations de grande taille, l'optimisation est souvent non compatible avec des temps de calcul raisonnables. Il semble donc qu'estimer la capacité de généralisation des contrôleurs ne peut pas être effectuée sans considérer les moyens de réduire le nombre d'évaluations du processus d'optimisation

De nombre méthodes ont été proposées pour diminuer le nombre d'évaluations d'un algorithme évolutionniste, notamment les procédures de *racing* [Birattari *et al.* 2002, Heidrich-Meisner & Igel 2009] et de *early stopping* [Bongard & Hornby 2010, Bongard 2011a]. La procédure de *racing* [Birattari *et al.* 2002, Heidrich-Meisner & Igel 2009] consiste à évaluer plusieurs fois les individus à l'aide d'un processus stochastique. Une solution sera rejetée dès que sa distribution de performances sera jugée significativement plus mauvaise que celles des autres solutions. L'étape de *racing* s'arrête une fois que toutes les solutions ont pu être classées (de la plus mauvaise à la meilleure) ou après un certain nombre d'évaluations. La technique de *early stopping* [Bongard & Hornby 2010, Bongard 2011a] est similaire dans le sens où l'évaluation d'une solution est stoppée si continuer l'évaluation ne lui permet pas de ne pas être dominée par une autre solution. Pour illustrer, imaginons deux solutions évaluées sur deux problèmes à la suite. Le premier problème est évalué sur 60 points et le second sur 40. Supposons que la solution 1 ait obtenu 60 points sur le premier problème et la solution 2 seulement 10 points. Quelque soit la note de la solution 2 sur le second problème, sa note globale ne dépassera pas celle de la solution 1 : on peut donc stopper l'évaluation de la solution 2. Ces approches font néanmoins des hypothèses sur la fonction de fitness utilisée, que ce soit en terme de distribution statistique pour le *racing* ou de monotonie pour le *early stopping*.

Une méthode alternative classiquement utilisée pour l'optimisation de fonction de fitness coûteuse en temps de calcul revient à construire un modèle de substitution (*surrogate model*) de cette fonction : les solutions sont alors évaluées pour une valeur approchée rapidement calculable avec le modèle et non pas une valeur exacte [Ratle 1998, Hemker *et al.* 2006, Hemker *et al.* 2009]. Ce type d'approche peut être utilisé pour construire une approximation de la performance des contrôleurs sur l'ensemble Ω_{eval}. L'utilisation de ces modèles de substitution sera détaillée dans le chapitre 3

2.4.3 Des robots qui s'adaptent

La plupart des travaux visant à optimiser des contrôleurs pour des robots partent du principe que l'environnement reste fixe et cherchent donc un comportement efficace pour cet environnement donné [Minato & Asada 2000]. Si le robot passe dans

un nouvel environnement, il ne lui sera donc pas possible d'adapter son comportement correctement et il continuera d'exploiter un comportement devenu sous-optimal, voire complètement inefficace. Un troisième problème important en robotique concerne donc l'adaptation du comportement du robot à une nouvelle situation, que ce soit un changement environnemental ou une modification de sa morphologie. Plus particulièrement, comment un robot dont le comportement est adapté à une situation donnée peut-il modifier son comportement quand il est confronté à une nouvelle situation ou à des pannes ? Le processus d'adaptation d'un robot à la nouvelle situation peut être divisé en trois phases [Bongard *et al.* 2006].

1. Le robot détecte que l'environnement a changé ou que son comportement actuel n'est plus suffisamment efficace.
2. Le robot entre dans une phase d'adaptation, pendant laquelle il cherche des comportements efficaces dans le nouvel environnement.
3. Un nouveau comportement est sélectionné et exploité tant qu'aucune nouvelle situation n'est détectée.

Nous nous focaliserons ici sur la deuxième étape de ce processus : la phase d'adaptation. On peut distinguer deux catégories de mécanisme d'adaptation dans la littérature, suivant que la méthode proposée assure juste un choix parmi plusieurs comportements pré-appris dans des situations pré-déterminées (e.g. [Weingarten *et al.* 2004, Steingrube *et al.* 2010]) ou qu'elle découvre effectivement un nouveau comportement, typiquement par un processus d'apprentissage par renforcement [Schaal & Atkeson 1994, Kohl & Stone 2004, Christensen *et al.* 2010] ou par optimisation (par exemple via un algorithme évolutionniste, e.g. [Bongard *et al.* 2006]).

Si les différentes situations que le robot est susceptible de rencontrer sont connues à l'avance, il est possible de définir un ensemble de comportements qui vont *a priori* fonctionner dans chacune des situations. Le robot doit juste apprendre à associer un ou plusieurs comportements à chaque situation. Un exemple de cette approche sur un robot mobile hexapode à 18 degrés de liberté est présenté dans l'article [Steingrube *et al.* 2010]. Dans ce travail, le robot est muni de 18 capteurs et il peut exploiter 11 comportements basiques (marche lente/rapide, marche quadrupède, se débloquer d'un trou, fuite, repos, ...). Un réseau de neurones basé sur un circuit CPG (Central Pattern Generator) est utilisé pour apprendre quelle combinaison de comportements sélectionner dans telle ou telle situation sensorielle. Cet apprentissage permet au robot d'adapter son comportement suivant l'environnement dans lequel il se trouve. Il est également possible de chercher un unique contrôleur permettant de s'adapter à un ensemble de situations pré-déterminées. Par exemple, Weingarten et al. [Weingarten *et al.* 2004] proposent de découper l'environnement du robot en plusieurs zones qui correspondent à différentes parties de la trajectoire optimale attendue. Le robot apprend alors à adapter son comportement dans chaque zone à l'aide d'un processus d'optimisation directement mené sur le robot. Il ne s'agit pas à proprement parler d'une sélection parmi plusieurs comportements pré-appris, mais le découpage en zones se fait en fonction d'un comportement optimal désiré

et aucun comportement original ne peut être découvert. Les travaux de Grand et al. [Grand *et al.* 2010], portant sur l'adaptation de posture du robot quadrupède Hylos à locomotion hybride roue-pattes [Grand *et al.* 2004b, Jarrault *et al.* 2010], rentrent également dans cette catégorie : le contrôleur du robot est ici construit de façon à ce que le robot adapte automatiquement sa posture sur un terrain accidenté pour maintenir son corps aussi horizontal que possible. Le mécanisme d'adaptation est alors intégré directement dans le contrôleur en tant que fonction "pré-apprise" des entrées sensorielles du robot. Bien qu'efficaces dans le cas où le robot ne peut rencontrer qu'un ensemble de situations limitées, ces différentes méthodes ne devraient toutefois pas toujours fonctionner correctement si le robot rencontre une situation complètement nouvelle non prévue par l'utilisateur. Dans ce cas, si aucun des comportements pré-appris n'est efficace, le robot n'aura pas la possibilité de découvrir un comportement original lui permettant de s'adapter correctement.

La seconde catégorie de mécanismes d'adaptation regroupe les travaux où le robot apprend un nouveau comportement efficace par lui-même, sans *a priori* sur la situation dans laquelle il se trouve et sur le comportement à découvrir. Cet apprentissage se fait classiquement par des techniques d'apprentissage par renforcement (à l'aide d'un modèle du robot [Schaal & Atkeson 1994] ou sans modèle par recherches locales de politique [Kohl & Stone 2004, Christensen *et al.* 2010]) ou par des méthodes d'optimisation (méthode de Powell [Sproewitz *et al.* 2008], algorithmes évolutionnistes [Bongard *et al.* 2006], ...). Ces approches nécessitent souvent un nombre important d'évaluations de contrôleurs avant de trouver un comportement efficace. Effectuer l'apprentissage directement sur le robot peut donc rendre la phase d'adaptation lente : par exemple, 3 heures pour apprendre un comportement de marche quadrupède sur un robot Sony AIBO [Kohl & Stone 2004] ou environ 20 minutes dans le cas d'un robot serpent [Sproewitz *et al.* 2008]. Ce coût expérimental est difficilement évitable quand aucun modèle précis du robot n'est pas disponible [Weingarten *et al.* 2004] et, même dans ce cas, les comportements optimaux d'après le modèle peuvent mal se transférer sur le robot physique. Une alternative est de modifier le comportement appris dans l'environnement précédent seulement là où il échoue dans le nouvel environnement [Minato & Asada 2000], conservant ainsi le reste du comportement, encore potentiellement efficace. Néanmoins, dans le cas où l'ancien comportement appris par le robot est complètement inefficace dans la nouvelle situation, une optimisation partant de zéro est de toute façon nécessaire.

Une alternative est que le robot construise un modèle de soi ou *self-model* à chaque fois qu'il se retrouve dans une nouvelle situation ou que son comportement n'est plus adapté. Pour ce faire, il est possible d'utiliser l'algorithme d'Estimation-Exploration [Bongard & Lipson 2005] qui se décompose en deux étapes : (1) un ou plusieurs modèles de soi du robot sont construits à partir des données déjà connues en réalité ; (2) on cherche les nouvelles expériences à effectuer sur le robot pour améliorer le ou les modèles de soi. Ces étapes peuvent être itérées jusqu'à ce qu'un assez bon modèle soit disponible. Le modèle de soi du robot peut ensuite être utilisé en tant que simulateur pour construire un contrôleur qui sera ensuite exploité par

le robot. Cette approche a par exemple été appliquée à une expérience de robotique résiliente, où un robot quadrupède doit s'adapter à un changement de morphologie [Bongard *et al.* 2006] (perte d'une patte). Le modèle de la morphologie du robot est tout d'abord identifié via l'algorithme d'Estimation-Exploration, dans l'optique de trouver un contrôleur de marche une fois qu'un modèle de soi suffisamment précis a été obtenu. Dans une seconde partie, le robot "perd" une patte. Le même processus est appliqué - identification du nouveau modèle de morphologie, puis construction d'un contrôleur de marche à partir du modèle de soi -, ce qui permet au robot de découvrir un mode de locomotion adapté à la nouvelle morphologie. Cette approche a l'avantage d'être plutôt rapide : environ 15 expériences sur le robot sont nécessaires pour trouver un modèle de soi dans l'expérience décrite. Toutefois, l'approche nécessite que le robot ré-apprenne de zéro un modèle de soi à chaque fois qu'une incohérence est détectée entre ce modèle et la réalité. Cela semble justifié dans le cas d'une altération de la morphologie du robot qui aura souvent une grande influence sur sa dynamique. Néanmoins, quand l'incohérence vient d'un changement de l'environnement, il est vraisemblable que l'ancien modèle soit encore relativement fiable sans qu'une nouvelle phase d'apprentissage ne soit nécessaire. D'autre part, la construction du modèle de soi n'est pas biaisée vers les zones comportementales intéressantes et pourrait nécessiter un nombre élevé d'expériences sur le robot dans d'autres applications.

Nous verrons dans la partie 6 que, si le robot dispose d'un modèle de soi, il est possible de reformuler la phase d'adaptation à un nouvel environnement comme un problème de transfert du modèle de soi à la réalité. Ceci nous permet, à partir de nos travaux sur le problème de la transférabilité, de définir un algorithme d'adaptation général et rapide.

2.4.4 Un problème commun : le temps d'évaluation

Au regard de la littérature, ces trois questions de robotique évolutionniste posent le problème du temps d'évaluation : il y a toujours un compromis entre le coût de l'évaluation dans l'environnement-cible (le robot ou l'ensemble des contextes d'évaluation) et la précision de l'évaluation dans un environnement simplifié (une simulation ou l'ensemble des contextes d'apprentissage). Mon approche générale se base sur les deux aspects suivants.

- L'optimisation doit essentiellement optimiser dans l'environnement simplifié afin d'avoir un coût d'évaluation faible.
- Le processus doit également prendre en compte des informations issues d'évaluations dans l'environnement-cible.

Pour ce second point, il n'est pas possible d'évaluer chaque solution dans l'environnement-cible. Si on s'autorise néanmoins à en évaluer quelques-unes, n'est-il pas possible d'en déduire des informations sur le comportement d'autres solutions dans l'environnement-cible ? En supposant que deux solutions aient des comportements proches dans l'environnement simplifié, il semble assez probable qu'elles auront des comportements relativement proches dans l'environnement-cible, menant à

des performances du même ordre. Ainsi, en calculant la performance d'une seule de ces solutions dans l'environnement-cible (donc en l'évaluant), il semble intuitivement possible de calculer une valeur de performance approchée pour l'autre solution.

Ce mécanisme d'approximation pour alléger le coût computationnel d'un processus d'optimisation est typiquement assuré en ingénierie par la construction d'un modèle de substitution qui, dans cet exemple, modéliserait la relation entre le comportement des solutions dans l'environnement simplifié et leur performance dans l'environnement-cible. Avant d'entamer une réponse aux trois questions posées précédemment, nous allons donc tout d'abord nous intéresser aux modèles de substitution et à leur utilisabilité dans un processus d'optimisation.

Chapitre 3

Modèles de substitution pour l'optimisation

Les modèles de substitution (ou *surrogate models* en anglais [1]) sont utilisés pour approximer la valeur d'une fonction inconnue (par exemple, la concentration en minerai), en fonction de plusieurs variables d'entrée (les coordonnées géographiques). Le but premier des modèles de substitution est de fournir une approximation de bonne qualité de la fonction inconnue en se basant sur un minimum de données expérimentales [Sacks *et al.* 1989, Barton 1998, Jones *et al.* 1998, Keane & Nair 2005, Jin 2005]. Typiquement, l'emploi de ces modèles intervient quand une évaluation exhaustive des valeurs intéressantes de la fonction inconnue serait trop longue ou coûteuse : par exemple, en géostatistiques pour connaître les caractéristiques d'un sol [Malvić & Dureković 2003, Hua *et al.* 2009], en aérospatiale pour évaluer l'impact de perturbations sur des matériaux [Simpson *et al.* 2001, Mack *et al.* 2007], en robotique pour modéliser ou générer la dynamique d'un robot [Kim *et al.* 2006, Hemker *et al.* 2009], ...

Un modèle de substitution peut être construit afin de fournir une bonne approximation globale du phénomène étudié [Sacks *et al.* 1989, Simpson *et al.* 2001, Malvić & Dureković 2003, Queipo *et al.* 2005, Kim *et al.* 2006]. Dans cette optique, le modèle obtenu doit être performant sur l'ensemble de l'espace d'entrée. Il peut ensuite être utilisé comme un simulateur rapide qui se substitue au phénomène réel. Nous nous intéresserons essentiellement ici à un autre type d'applications [Jones *et al.* 1998, Jones 2001, Jin 2005, Hemker *et al.* 2009], où le modèle de substitution est construit au cours d'un processus d'optimisation en tant que fonction de performance à optimiser. Dans ce cas, le modèle cherche à approximer la performance des solutions en se basant sur un ensemble de valeurs exactes de la fonction inconnue. Le modèle est amélioré au cours de l'optimisation par l'évaluation de nouveaux points sélectionnées suivant une stratégie de mise à jour donnée.

Le but premier qui justifie l'emploi d'un modèle de substitution dans un cadre d'optimisation est de réduire le nombre d'évaluations exactes de solutions pendant le processus d'optimisation. Il n'y a pas de consensus dans la littérature sur le type de modèles à utiliser ou sur la stratégie de mise à jour à employer. Le but de cette section n'est pas d'en faire la liste exhaustive, mais de présenter les principaux types de modèles de substitution existants, les hypothèses éventuelles sous-jacentes à leur

1. Suivant le domaine, on peut aussi rencontrer les termes *response surface models* ou *metamodels*.

utilisation, ainsi qu'un bref aperçu de leur utilisabilité dans le cadre d'un processus d'optimisation.

3.1 Modèle de substitution et interpolation

Le but premier d'un modèle de substitution est, comme son nom l'indique, de se substituer à une fonction exacte. En pratique, de telles techniques sont utilisées quand les points de la fonction exacte sont coûteux à évaluer, par exemple en terme de temps expérimental.

Supposons que l'on cherche à optimiser la conception d'une aile d'avion. Construire une aile dépend de nombreux paramètres : les matériaux utilisés, la géométrie de l'aile, sa courbure, ... Evaluer de manière exacte une solution (i.e. une combinaison des différents paramètres) nécessite ici de construire l'aile et de la soumettre à différents tests aérodynamiques pour estimer sa performance. Dans cet exemple, l'évaluation d'un jeu de paramètres va être à la fois longue et coûteuse et on ne pourra vraisemblablement pas s'autoriser l'évaluation exacte de toutes les solutions potentielles.

Imaginons maintenant que nous ayons déjà un certain nombre de jeux de paramètres pour lesquelles la performance de l'aile correspondante soit disponible (typiquement les modèles d'aile déjà existants). Il est alors possible de construire un modèle approché de la fonction reliant les paramètres de l'aile et le critère de performance en se basant uniquement sur ces points déjà connus. Une fois le modèle construit, il permet d'obtenir une valeur de performance approchée pour toute solution. Il est alors par exemple possible d'optimiser les paramètres de l'aile à partir de ce modèle approché qui se substitue à la fonction de performance exacte, d'où son nom de modèle de substitution.

Dans cette partie, nous allons passer rapidement en revue les différentes techniques qui permettent, à partir de points de données connus, de construire un modèle de substitution avant de s'attarder plus en détail sur les principales structures de modèles qui sont habituellement utilisés.

3.1.1 Construction du modèle de substitution

3.1.1.1 Principes de la régression

La construction d'un modèle de substitution est typiquement un problème de régression à partir d'un ensemble de données générées sur le système étudié [Jin 2005, Kim *et al.* 2006]. La régression regroupe l'ensemble des méthodes qui visent à modéliser et analyser la relation entre une variable observée y et une ou plusieurs variables d'entrées notées $X = (x_1,\ x_2,\ \ldots,\ x_n)$. Le modèle est obtenu en se basant sur un ensemble de n points de donnée déjà connus de la forme $p^i = (X^i,\ y^i)$ avec $0 < i \leq n$. On peut distinguer deux types de régression :

- la régression paramétrique où la structure du modèle est fixée par l'utilisateur et où seul un ensemble de paramètres reste à estimer à partir des données pour

connaître le modèle ;
- la régression non paramétrique où à la fois la structure et les éventuels paramètres sont estimés à partir des données, ce qui peut nécessiter de connaître beaucoup de données sur le système étudié.

Une fois que la structure du modèle a été fixée, et soit m le nombre de paramètres variables de cette structure, trouver le modèle de substitution $\mathcal{M}$ qui modélise au mieux les données revient à chercher le vecteur de paramètres qui vérifie un critère donné. Les méthodes de régression cherchent classiquement le vecteur de paramètres qui minimise un critère d'erreur E entre les valeurs prédites par le modèle $\mathcal{M}$ aux points p_i et les valeurs exactes en ces points :

$$E = \sum_{i \in [1,\ n]} ||y^i - \mathcal{M}(X^i)||^2$$

Dans le cadre de nombreux travaux utilisant des algoritmes évolutionnistes, des modèles de substitution obtenus par régression sont utilisés pour construire un modèle d'une fonction de fitness coûteuse à évaluer [Jin 2005]. Les principaux types de modèles employés sont :
- les modèles polynômiaux ;
- les réseaux de neurones (perceptron multi-couches ou réseau de fonctions à base radiale) ;
- les processus gaussiens ou modèles de Kriging ;
- les machines à vecteurs de support.

Sans entrer dans les détails, l'étape d'estimation des paramètres de ces différents modèles et leur utilisation comme prédicteur peuvent être coûteuse en temps de calcul, notamment si la dimension de l'espace d'entrée est élevée. Dans nos travaux, nous avons fait le choix d'utiliser un cas particulier des méthodes de régression : les méthodes d'interpolation. En effet, à notre connaissance, les nombreux travaux de la littérature n'ont pour l'instant pas permis de déterminer un meilleur type de modèle, que ce soit parmi les modèles de régression ou en comparant modèles de régression et modèles d'interpolation (voir par exemple [Barthelemy *et al.* 1994, Giunta *et al.* 1998, Jin *et al.* 2001, Mullur & Messac 2006]). D'autre part, le choix d'une méthode de régression spécifique n'est a priori pas crucial dans ce manuscrit et nous avons ainsi arbitrairement sélectionné l'une des méthodes d'interpolation les plus simples et les moins coûteuses, l'interpolation par *Inverse Distance Weighting*.

3.1.1.2 Cas particulier de l'interpolation

L'interpolation [Giunta *et al.* 1998, Simpson *et al.* 2001] est un cas particulier de la régression où, pour trouver le vecteur de paramètres optimaux, on contraint le modèle $\mathcal{M}$ à prédire de manière exacte les points connus :

$$\forall i \in [1,\ n],\ \mathcal{M}(X^i) = y^i$$

Dans ce cas, le nombre de paramètres m à estimer est typiquement égal au

nombre de points connus n. Il existe alors un unique vecteur de paramètres tel que la prédiction du modèle aux points connus soit exacte.

3.1.1.3 Choix d'une méthode de construction

Pour construire un modèle de substitution, on peut utiliser des méthodes de régression paramétriques ou non paramétriques. Néanmoins, la détermination de la structure d'un modèle par des méthodes non paramétriques peut requérir un nombre important de données expérimentales sur le système étudié. Nous n'allons donc considérer ici que des méthodes de régression paramétriques.

Dans ce contexte, la structure du modèle sera fixée *a priori* par l'utilisateur et seuls les paramètres variables du modèle doivent être déterminés. Le choix entre des méthodes d'interpolation ou des méthodes de régression pour construire un modèle de substitution est une question ouverte qui n'a à notre connaissance que peu été abordée dans la littérature. Certains auteurs utilisent des méthodes d'interpolation (e.g. [Giunta *et al.* 1998] et [Simpson *et al.* 2001]), mais d'autres travaux sont basés sur des méthodes de régression plus générales (e.g. [Jin 2005] et [Kim *et al.* 2006]).

Les méthodes de régression paraissent plus adaptées aux situations où la variable observée est bruitée et où l'on dispose de plusieurs estimations de la foncion par point de données. D'autre part, le nombre de données doit être suffisamment important par rapport au nombre de paramètres de la structure de modèle choisi pour assurer une bonne estimation des paramètres. D'autre part, en choisissant le vecteur de paramètres qui minimise un critère d'erreur sur l'ensemble des points connus utilisés (cf. partie 3.1.1.1), les méthodes de régression permettent de construire un modèle correct en moyenne mais sans garantie sur l'erreur de prédiction en un point connu donné : l'écart entre le modèle et la valeur connue en un point peut être important.

Au contraire, les méthodes d'interpolation assurent par construction que l'erreur commise par le modèle sur les points connus est nulle. Nous pensons que cette propriété peut être critique pour un modèle de subsitution construit pendant un processus d'optimisation, en particulier si la méthode de mise à jour du modèle consiste à tester sur le système étudié des points de donnée potentiellement optimaux, par exemple comme c'est typiquement le cas dans des processus d'optimisation basés sur les modèles de Kriging (e.g. [Jones 2001]). Le modèle qui prend en compte un nouveau point doit en effet contenir l'information suivante : le point testé est-il oui ou non meilleur que les points déjà connus ? En contrepartie, utiliser une méthode d'interpolation requiert de faire l'hypothèse que les points de donnée sont peu ou pas bruités et donc que les valeurs de la variable observée sont connues de manière exacte.

Les modèles construits par une méthode d'interpolation sont souvent des agrégations de modèles locaux (voir par exemple [Shepard 1968] et [Buhmann 2000]), ce qui peut introduire des optima locaux : si le modèle de substitution est utilisé à des fins d'optimisation, la présence de tels optima locaux risque d'entraver le processus d'optimisation. Suivant la structure de modèle choisie, les méthodes de régression plus générales peuvent être moins sensibles à ce problème.

D'autre part, un autre problème concerne le phénomène de Runge [Runge 1901] principalement connu dans le cas de modèles d'interpolation polynomiale (plus de détails dans la partie 3.1.2.1 ; voir figure 3.1 pour une illustration). Ce phénomène intervient quand un modèle obtenu par interpolation prédit de fortes oscillations entre les points connus utilisés pour estimer les valeurs de paramètres. D'après la littérature, ce phénomène semble apparaître essentiellement dans le cas où les points connus sont équirépartis dans l'espace d'entrée du modèle (e.g. [Epperson 1987, Fornberg & Zuev 2007]) ce qui ne sera jamais le cas dans nos expériences.

Il est vraisemblable que, dans ce travail, le choix de la méthode de construction du modèle de substitution ne soit pas critique. Nous avons choisi d'utiliser des méthodes d'interpolation, car les valeurs de la variable observée sont, dans notre cas, sujettes à peu de variabilité. Les mesures sur les systèmes robotiques considérés dans nos différents protocoles sont en effet effectuées à l'aide de systèmes de capture de mouvement précis (de l'ordre du millimètre en 3D avec un système CODA cx1 [2]). De plus, dans nos différentes expériences, la variabilité de comportements obtenus avec un même contrôleur est faible.

3.1.2 Méthodes d'interpolation

Tout modèle construit par interpolation vérifie la contrainte suivante : le modèle renvoie la valeur exacte de la fonction à approximer sur les n points qui ont permis de le construire. Soit $\mathbb{P}$ l'ensemble des points utilisés pour construire le modèle $\mathcal{M}$ de la fonction $\mathcal{F}$, on a :

$$\forall p \in \mathbb{P},\ \mathcal{M}(p) = \mathcal{F}(p).$$

Cette contrainte n'est vérifiable que si la fonction $\mathcal{F}$ est déterministe, c'est-à-dire que chaque point p correspond à une valeur unique de $\mathcal{F}$, notée $\mathcal{F}(p)$. En pratique, si le bruit est assez faible, les méthodes d'interpolation permettront tout de même d'obtenir de bonnes approximations de la fonction.

Notations :

- $\mathbb{X}$: l'ensemble des points possibles
- $\mathbb{P}$: l'ensemble des points utilisés pour construire le modèle d'interpolation $\mathcal{M}$ de la fonction $\mathcal{F}$ ($\mathbb{P} \subset \mathbb{X}$)

3.1.2.1 Interpolation polynomiale ou par splines

L'interpolation polynomiale revient à approcher une fonction dont $n+1$ points distincts sont connus par un polynôme $\mathcal{M}$ de degré n.

$$\begin{cases} \forall x \in \mathbb{X}, & \mathcal{M}(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \\ \forall p \in \mathbb{P}, & \mathcal{M}(p) = \mathcal{F}(p) \end{cases}$$

2. Charnwoord Dynamics Ltd, UK

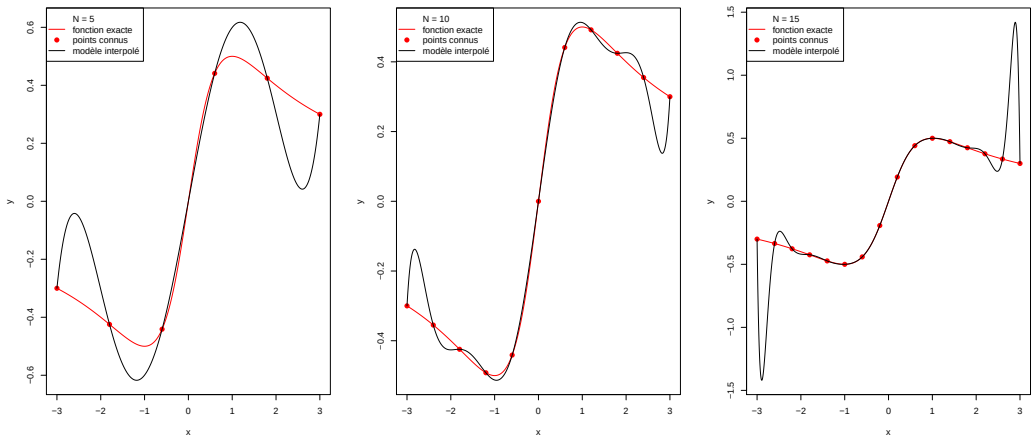

FIGURE 3.1 – Illustration du phénomène de Runge lors de l'interpolation de la fonction $\mathcal{F}(x) = \frac{x}{1+x^2}$ sur l'intervalle $[-3, 3]$.

La contrainte d'interpolation permet de trouver les coefficients $a_i,\ i \in [0,\ n]$ par résolution d'un système d'équations linéaires. On peut démontrer qu'il n'existe qu'un unique polynôme qui vérifie cette contrainte. Pour trouver les valeurs des $n+1$ coefficients, il est nécessaire de connaître la valeur de la fonction $\mathcal{F}$ en $n+1$ points distincts.

Cette méthode est généralisable pour des dimensions d'entrée quelconque. Par exemple, si $\mathcal{F}$ est une fonction de deux variables x_1 et x_2, on cherchera à trouver les coefficients du polynôme suivant :

$$\mathcal{M}(x_1,\ x_2) = c_{00} + c_{10}x_1 + c_{01}x_2 + c_{11}x_1x_2 + c_{20}x_1^2 + c_{02}x_2^2 + \ldots + c_{n_1n_2}x_1^{n_1}x_2^{n_2}$$

Pour identifier les $(n_1+1)\times(n_2+1)$ coefficients du polynôme, il est nécessaire de connaître $(n_1+1)\times(n_2+1)$ points de la fonction, donc potentiellement beaucoup de points suivant le degré du polynôme. En pratique, on emploie plutôt souvent un polynôme plus simple dit bilinéaire qui nécessite de connaître 4 points de la fonction :

$$\mathcal{M}(x_1,\ x_2) = c_{00} + c_{10}x + c_{01}y + c_{11}xy$$

Le choix du degré du polynôme dépend totalement de l'utilisateur. Intuitivement, on pourrait penser qu'un degré plus élevé mène à des modèles de meilleure qualité, étant donné qu'ils sont basés sur un plus grand nombre de points connus et peuvent capter des formes plus compliquées. En pratique, un degré n élevé rend la résolution du système d'équations pour trouver les coefficients plus difficile. Dans certains cas, il est même possible qu'augmenter le degré du polynôme entraîne une baisse de performance générale du modèle. Un cas particulier est le phénomène de

Runge qui peut être observé si les points connus sont équidistants dans l'espace. Ce phénomène, illustré sur la figure 3.1 avec une seule variable d'entrée, entraîne le polynôme interpolé à osciller entre les points d'interpolation, conduisant à une approximation peu fiable.

Pour résoudre le problème posé par ce phénomène de Runge, on peut recourir à l'interpolation par splines qui revient à construire une fonction polynomiale par morceaux. En une dimension, cette méthode d'interpolation construit un polynôme local entre chaque pair de points connus consécutifs. Comme le degré des polynômes locaux est bas (classiquement égal à 3 dans le cas de l'interpolation par splines cubiques), le problème de Runge n'est pas observé. Par contre, cette méthode n'est généralisable en dimensions plus élevées que si les points connus sont sur une grille régulière dans l'espace d'entrée.

3.1.3 Interpolation par *Inverse Distance Weighting*

L'interpolation par *Inverse Distance Weighting* [Shepard 1968] ou IDW est basé sur l'hypothèse que la valeur de la fonction en un point p dépend plus des valeurs des points connus proches de p dans l'espace d'entrée que des valeurs des points éloignés de p. La contribution d'un point connu p_1 à la valeur approchée au point p dépend uniquement de la distance entre p_1 et p. Cette méthode a donc l'avantage d'être applicable quelle que soit la dimension du vecteur d'entrée sans introduire de paramètres supplémentaires et est une des méthodes d'interpolation les plus simples utilisées en géostatistiques. Le seul paramètre κ de l'interpolation par IDW définit la régularité de la fonction interpolée [Modalaldoust 2010] : plus sa valeur est élevée, plus l'interpolation aura des pentes abruptes. En pratique, le modèle $\mathcal{M}$ se construit comme suit :

$$\begin{cases} \forall x \in \mathbb{X}, \quad \mathcal{M}(x) = \left[\sum_{p\in\mathbb{P}} \frac{\mathcal{F}(p)}{\|x-p\|^\kappa}\right] \cdot \left[\sum_{p\in\mathbb{P}} \frac{1}{\|x-p\|^\kappa}\right]^{-1} \\ \forall p \in \mathbb{P}, \quad \mathcal{M}(p) = \mathcal{F}(p) \end{cases}$$

Pour prendre en compte un nouveau point connu dans le modèle, il suffit d'ajouter un terme supplémentaire sans modification du reste du modèle. La principale limitation de cette méthode d'interpolation réside dans le fait que la valeur prédite se situe nécessairement entre le minimum et le maximum des valeurs connues utilisées : elle ne prend pas en compte les informations de pente de la fonction. En contrepartie, l'interpolation est rapide à calculer et ne dépend d'aucun coefficient ou autre paramètre de pondération.

Dans sa version la plus simple, l'interpolation par IDW prédit la valeur d'un point en utilisant les valeurs de tous les points connus, même les plus éloignés. Bien que la pondération par la distance favorise effectivement les valeurs des points proches des valeurs des points éloignés, ces termes peuvent avoir une influence non négligeable sur la valeur prédite, notamment si le nombre de points utilisés est élevé

ou si le paramètre κ est petit. L'influence du paramètre κ est visible sur les deux figures 3.3 et 3.6 dans deux cas simples d'approximation de fonction.

Plusieurs améliorations ont été proposées pour aborder ce problème, par exemple en ne prenant en compte que les points voisins (dont la distance est inférieure à un seuil), ou encore en utilisant les k plus proches voisins [Li & Chan Hilton 2007]. Néanmoins, ces approches introduisent des paramètres additionnels qui ont une grande influence sur l'interpolation.

La méthode par IDW est classiquement utilisée en géostatistiques, mais son utilisation en optimisation est rare, et ce malgré sa simplicité. De nombreux travaux utilisent les méthodes de *kriging* que nous présenterons plus tard et qui sont également issues du domaine des géostatistiques.

3.1.4 Interpolation par réseaux de fonctions à base radiale

Une fonction à base radiale ou RBF est une fonction non-linéaire ϕ dont la valeur ne dépend que de la distance entre le point considéré et un point de référence c appelé centre. Nous noterons la valeur au point x de cette fonction $\phi_c(x) = \phi(\|x - c\|)$.

De telles fonctions peuvent être agrégées en une somme pondérée sous forme d'un réseau de fonctions à base radiale et ainsi servir de modèle d'interpolation [Buhmann 1993, Dyn & Ron 1995, Buhmann 2000]. Les centres sont choisis parmi les points de données connus. En supposant qu'on souhaite interpoler en se basant sur les n points connus $c_i \in \mathbb{P}$, $i \in [1,\ n]$, le réseau de RBF construit par interpolation s'écrit :

$$\begin{cases} \forall x \in \mathbb{X}, \quad \mathcal{M}(x) = \displaystyle\sum_{i \in [1,\ n]} w_i\ \phi(\|x - c_i\|) \\ \\ \forall p \in \mathbb{P}, \quad \mathcal{M}(p) = \mathcal{F}(p) \end{cases}$$

Les poids w_i de chaque RBF sont calculés entre résolvant un système d'équations linéaires $\mathbf{A} \cdot \mathbf{w} = \mathbf{b}$, où $\mathbf{A}$ désigne la matrice de dimensions $n \times n$ contenant les distances centre à centre déformées par la fonction ϕ et où $\mathbf{b}$ est le vecteur de taille n contenant les valeurs de la fonction utilisées pour l'interpolation :

$$\begin{bmatrix} \phi_{c_1}(c_1) & \phi_{c_1}(c_2) & \dots & \phi_{c_1}(c_n) \\ \phi_{c_2}(c_1) & \phi_{c_2}(c_2) & \dots & \phi_{c_2}(c_n) \\ \vdots & & \ddots & \vdots \\ \phi_{c_n}(c_1) & \phi_{c_n}(c_2) & \dots & \phi_{c_n}(c_n) \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \mathcal{F}(c_1) \\ \mathcal{F}(c_2) \\ \vdots \\ \mathcal{F}(c_n) \end{bmatrix}$$

Ce système peut être résolu de manière exacte si la matrice $\mathbf{A}$ est inversible, ce qui est le cas pour la plupart des fonctions ϕ utilisées. Classiquement, le réseau est constitué de fonction à base radiale gaussienne de la forme suivante :

$$\phi_c(x) = \exp(-\frac{\|x - c\|^2}{2\ \sigma^2})$$

Ce type de fonction gaussienne est locale dans le sens où si la distance entre x et c est grande, $\phi_c(x)$ est faible, voire négligeable. La constante σ permet de régler la pente de la fonction : plus *sigma* est petit, plus la pente est forte et moins la RBF contribue aux valeurs des points éloignés. D'autres RBF peuvent être utilisées, éventuellement non locales, mais les caractéristiques de la fonction interpolée sont assez robustes indépendamment du type de fonction utilisé [Bishop 1995].

Comme l'interpolation par IDW, ce type de modèle de substitution ne dépend pas du nombre de variable d'entrées de la fonction $\mathcal{F}$, mais seulement de la fonction de distance définie sur ces variables. Par contre, le nombre de points connus a une influence directe sur le nombre de paramètres de pondération à calculer : à chaque point connu correspond une fonction radiale et donc un poids. En conséquence, si l'interpolation se base sur beaucoup de points, le nombre de paramètres va également être important, conduisant à une inversion d'une matrice $\mathbf{A}$ de grande taille pour trouver leurs valeurs. D'autre part, la prise en compte d'un nouveau point dans le modèle d'interpolation entraîne la mise à jour de l'ensemble des poids.

3.1.5 Interpolation par *kriging*

Les méthodes de *kriging* sont initialement des techniques d'interpolation issues des géostatistiques [Matheron 1963, Krige 1981, Cressie 1990], utilisées par exemple pour prédire l'altitude d'un relief ou la concentration en minerai d'un terrain. Le modèle utilisé est basé sur un prédicteur linéaire en fonction des données connues qui prend en compte l'espérance mathématique, ainsi que la covariance de la variable à interpoler. Une fois le modèle construit, il est aussi possible de construire un prédicteur de l'erreur commise, notamment utilisable pour trouver les points de données qui amélioreraient le plus le modèle s'ils étaient connus. Les méthodes de *kriging* sont sans doute à l'heure actuelle les techniques d'interpolation les plus utilisées, aussi bien en optimisation qu'en géostatistiques ou encore en météorologie [Azimi-Zonooz *et al.* 1989, Cressie 1990, Malvić & Đureković 2003, Mueller *et al.* 2004, Jin 2005, Hemker *et al.* 2009, Soenario & Sluiter 2010].

Dans la suite de cette partie, nous supposerons que la fonction à interpoler $\mathcal{F}$ est une fonction de $\mathbb{R}^e$ dans $\mathbb{R}$, avec e le nombre de variables de $\mathcal{F}$. Dans sa forme générale, le modèle de *kriging* cherche, comme l'interpolation par IDW, à prédire la valeur en un point à l'aide d'une combinaison linéaire des données connues. Le modèle peut se décomposer en deux termes, μ et δ. μ est le modèle global moyen qui peut être constant ou plus perfectionné suivant la méthode de *kriging* utilisée. L'écart local à μ est approximé par une fonction stochastique δ d'espérance nulle et de covariance non nulle qui suit une loi normale.

Soit $\mathcal{P} = \{p_1,\ p_2,\ \ldots,\ p_n\}$ l'ensemble des n points connus utilisés pour l'interpolation et $\mathcal{F}$ la fonction à interpoler, le modèle de *kriging* général $\mathcal{Z}$ peut s'écrire comme suit :

$$\mathcal{Z}(x) - \mu(x) = \delta(x) = \sum_{i\ \in\ [1,\ n]} w_i(x)\ [\mathcal{F}(p_i) - \mu(p_i)]$$

Suivant le même principe que l'interpolation par IDW, les méthodes de *kriging* sont basées sur l'hypothèse que la prédiction en un point dépendra plutôt des valeurs connues en des points proches. La construction d'un modèle de *kriging* passe par l'estimation du semi-variogramme, qui reflète la dépendance spatiale entre les points. Ce semi-variogramme est ensuite utilisé pour déterminer les poids $w_i(x)$. Plus précisément, le semi-variogramme se définit comme suit. Soit deux points $\boldsymbol{x}$ et $\boldsymbol{y}$, la valeur du semi-variogramme γ correspondante est :

$$\gamma(\boldsymbol{x},\ \boldsymbol{y}) = \frac{1}{2}\ \mathrm{Var}\left[\mathcal{Z}(\boldsymbol{x}) - \mathcal{Z}(\boldsymbol{y})\right]$$

Var [.] représente la fonction variance. Classiquement, les méthodes de *kriging* reposent sur l'hypothèse que, le processus modélisé $\mathcal{Z}$ est stationnaire et isotropique. Autrement dit, la valeur de γ ne dépend que de la distance entre les points $\boldsymbol{x}$ et $\boldsymbol{y}$ considérés. Si on cherche à modéliser des régions très hétérogènes, il est possible de découper l'espace en différentes zones où les hypothèses de stationnarité et d'isotropie sont vérifiées et d'y effectuer des interpolations locales.

On peut montrer que la connaissance du semi-variogramme γ est équivalente à celle de l'espérance mathématique du processus $\mu(x) = \mathrm{E}\left[\mathcal{Z}(x)\right]$ et de la covariance $\mathrm{cov}(\boldsymbol{x},\ \boldsymbol{y})$. Les différentes variantes d'interpolation par *kriging* se distinguent suivant le modèle utilisé pour construire $\mu(x)$:

- $\mu(x)$ est une constante connue pour la méthode de *kriging* simple ;
- $\mu(x)$ est une constante à estimer pour la méthode de *kriging* ordinaire ;
- $\mu(x)$ est un polynôme à estimer pour la méthode de *kriging* universel.

Il existe d'autres variantes où le modèle moyen μ est plus complexe, mais il s'avère que les méthodes les plus utilisées dans la littérature sont les trois variantes ci-dessus.

Par rapport aux méthodes d'interpolation précédentes, le modèle de *kriging* permet de calculer une fonction d'amélioration attendue : pour un point p de l'espace quelconque, cette fonction estime le gain de qualité entre le modèle courant et un modèle où p est ajouté aux points connus utilisés pour l'interpolation. Ainsi, plus la valeur d'amélioration attendue est forte, plus le point concerné se situe dans une zone potentiellement mal modélisée par le modèle. Cette fonction va donc être particulièrement intéressante dans le cas où le modèle est mis à jour au cours de l'optimisation et nous reviendrons sur les avantages et inconvénients d'utiliser cette fonction d'amélioration attendue dans la partie 3.3.

La différence essentielle entre les utilisations des méthodes de *kriging* en géostatistiques et en optimisation repose sur le fait que le semi-variogramme est estimé à l'aide de données du terrain en géostatistiques, alors qu'il est souvent arbitraire en optimisation [Sasena 2002]. Ce point est particulièrement important car les comparaisons entre les méthodes de *kriging* et des méthodes plus simples tendent à montrer que l'interpolation par *kriging* ne marche significativement mieux que si le semi-variogramme est suffisamment réaliste [Malvić & Dureković 2003, Mueller *et al.* 2004]. Dans le cas contraire, la méthode donne des résultats similaires, voire inférieurs à des techniques d'interpolation par IDW ou encore utilisant

des RBF.

En optimisation, le modèle de covariance utilisé est souvent basé sur une fonction exponentielle [Jin 2005, Jones 2001, Fu *et al.* 2008]. Soit $\boldsymbol{x}$ et $\boldsymbol{y}$ deux points, h_i la distance entre les $i^{ème}$ composantes de ces deux points, et e le nombre total de composantes (i.e. le nombre de variables de la fonction $\mathcal{F}$), la valeur de covariance entre $\boldsymbol{x}$ et $\boldsymbol{y}$ est :

$$\text{cov}(\boldsymbol{x},\ \boldsymbol{y}) = \sigma \prod_{i=1}^{e} \exp\left[-\theta_i\ h_i^{\alpha_i}\right]$$

Cette fonction nécessite de fixer les paramètres θ_i, σ et α_i, soit $2 \cdot e + 2$ paramètres au total. Ainsi, contrairement aux modèle d'interpolation présentés précédemment, la complexité d'un modèle de *kriging* va dépendre de la taille de l'espace d'entrée de $\mathcal{F}$. Dans certains cas, il est possible de fixer un certain nombre de ces paramètres *a priori* : par exemple, $\forall i \in [1,\ e]$, $\alpha_i = 2$ [Leary *et al.* 2004, Jin 2005, Hemker *et al.* 2006]. Le nombre de paramètres de covariance est alors $e + 2$.

En résumé, un modèle complet de *kriging* basé sur la fonction de covariance ci-dessus va compter au moins $n + e + 2$ paramètres : n poids correspondant aux n points connus utilisés et un nombre de paramètres linéaire en fonction de e pour la covariance. Ainsi, si le nombre de variables d'entrée de la fonction à interpoler est important, l'utilisation d'un modèle de *kriging* va nécessiter de connaître beaucoup de points avant de pouvoir être initialisé correctement.

3.1.6 Conclusion intermédiaire

Les différentes structures habituellement utilisées pour construire un modèle d'interpolation ont été présentées. Un point important est à prendre compte : le nombre de paramètres à estimer pour utiliser ces modèles. Le tableau 3.1 récapitule le nombre de paramètres total à fixer pour chaque technique d'interpolation dans le cas général. Un nombre plus important de paramètres entraîne nécessairement le besoin de connaître plus de points de la fonction $\mathcal{F}$ pour l'initialiser. Cette information est donc importante dans le cadre des modèles de substitution utilisés pour l'optimisation, étant donné que le but premier est de limiter au minimum le nombre d'évaluations de $\mathcal{F}$. Le choix d'un modèle élaboré mais introduisant des paramètres supplémentaires par rapport à un modèle simple devrait donc automatiquement être justifié par un gain systématique en terme de qualité de l'interpolation résultante.

La littérature compte un nombre élevé de comparaisons entre ces différentes méthodes dans des contextes plus ou moins appliqués. Il est difficile d'en tirer des conclusions générales, mais on peut mettre en évidence quatre points :

- les méthodes de *kriging* sont plus performantes que des interpolations par IDW et par RBF si le semi-variogramme est rigoureusement estimé [Hua *et al.* 2009, Soenario & Sluiter 2010] ;

Modèle d'interpolation	Poids		Méta-paramètres		Nombre total de paramètres à optim.
	fixés	optim.	fixés	optim.	
polynômes/splines		x	x		$\prod_{i \in [1,\ e]} (m_i + 1)$
IDW	x		x		0
RBF		x	x		n
kriging ordinaire		x		x	$n + e + 2$

n : nombre de points connus utilisés pour l'interpolation
e : nombre de variables d'entrées du modèle
m_i : degré maximal du polynôme selon l'entrée i, $i \in [1,\ e]$

TABLE 3.1 – Tableau récapitulatif du nombre et de l'origine des différents paramètres à fixer pour chaque technique d'interpolation.

- sans estimation précise du semi-variogramme, les méthodes de *kriging* ne fonctionnent pas significativement mieux que des méthodes plus simples, l'interpolation par IDW notamment [Malvić & Dureković 2003, Mueller *et al.* 2004] ;
- le modèle de *kriging* est basé sur une hypothèse de normalité des données et est donc sensible aux transformations sur les données [Jones *et al.* 1998, Hua *et al.* 2009] ;
- la méthode par IDW est la plus rapide [Witteveen & Bijl 2009] et ne dépend d'aucun paramètre [Shepard 1968] contrairement aux méthode par *kriging* et RBF.

3.2 Bref comparatif

Maintenant que différentes méthodes d'interpolation ont été présentées, il est intéressant de tester plus en détail leurs comportements respectifs sur des problèmes-jouets. Cette partie n'a pas pour but de déterminer quelle méthode est meilleure que telle autre, mais d'affirmer deux points :

- ces méthodes, bien que basées sur des structures de modèle différentes, donnent de bonnes interpolations ;
- les différences de résultats sont souvent liées à l'intégration par l'utilisateur d'information dans le modèle.

Pour montrer ces deux points, nous allons indiquer les résultats des différentes méthodes sur l'interpolation respective de deux fonctions classiquement utilisées comme test de méthodes d'optimisation [Storn & Price 1995, Alliot 1996, Tvrdik & Krivy 1999, Hansen *et al.* 2009] : la fonction de De Jong et la fonction de Rosenbrock. Nous allons comparer, pour chacune de ces fonctions, les caractéristiques des approximations obtenues avec diverses structures de modèles de substitution, toutes construites à partir d'un même ensemble de points connus fixés.

Modèle	Paramètres	MSE	nMSE	Corrélation
Inverse Distance Weighting (κ=2)	0	0.059	0.105	0.919
Inverse Distance Weighting (κ=4)	0	0.047	0.089	0.909
Radial Basis Functions	13	0.005	0.010	0.998
Ordinary kriging	13 + 4	0.016	0.008	0.996

TABLE 3.2 – Erreurs et corrélations entre la fonction initiale et l'approximation obtenue avec chaque type de modèle indiqué pour la fonction de De Jong.

3.2.1 Résultats sur la fonction de De Jong

La fonction de De Jong est une fonction à deux variables de $\mathbb{R}^2$ dans $\mathbb{R}$. Pour tout couple $(x,\ y)$, elle se calcule comme suit :

$$\mathcal{F}_1(x,\ y) = x^2 + y^2$$

Les 4 techniques d'interpolation suivantes ont été implémentées :

- interpolation par IDW avec $\kappa = 2$;
- interpolation par IDW avec $\kappa = 4$;
- interpolation par RBF ;
- interpolation par *kriging* ordinaire.

L'interpolation est réalisée à partir de 13 points répartis comme indiqué sur la figure 3.2. La surface interpolée obtenue dans chaque cas est indiquée sur la figure 3.3 : chaque ligne de la figure indique la fonction originale, la fonction interpolée et la surface d'erreur entre les deux fonctions. La figure 3.4 indique l'amélioration attendue prédite par le modèle de *kriging*.

Le tableau 3.2, pour chaque méthode, le nombre total de paramètres qu'il a fallu estimer, ainsi que l'erreur quadratique moyenne (MSE), l'erreur quadratique moyenne normalisée (nMSE) et le coefficient de corrélation de Pearson entre la fonction à interpoler et l'interpolation.

Pour avoir une bonne estimation de ces trois valeurs, elles sont calculées en échantillonnant les deux fonctions sur une grille régulière de $201 \times 201 = 40401$ points équirépartis dans l'espace d'entrée $[-1,\ 1]$. Les valeurs de la fonction sont dans l'intervalle $[0,\ 2]$

Pour ce premier test d'interpolation sur la fonction de De Jong, les méthodes par RBF et par *kriging* ordinaire obtiennent les meilleurs résultats avec des coefficients de corrélation supérieurs à 0.99 (nMSE $\simeq$ 0.01). L'interpolation par IDW est également performante avec des coefficients de corrélation supérieurs à 0.90 (nMSE $\simeq$ 0.10). L'influence du paramètre κ ne semble pas déterminante avec les valeurs utilisées (2 et 4).

Dans le cas de cette fonction très simple, les méthodes par RBF et par *kriging* ordinaire semblent donc meilleures. Il est néanmoins intéressant de noter que leur gain de performances par rapport à une interpolation par IDW nécessite l'estimation de 13 paramètres supplémentaires.

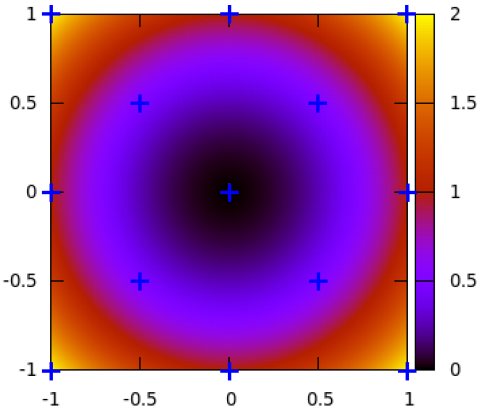

FIGURE 3.2 – Représentation de la fonction de De Jong avec $(x,\ y) \in [-1,\ 1]^2$ – Les croix bleues correspondent aux 13 points utilisés pour l'interpolation.

3.2.2 Résultats sur la fonction de Rosenbrock

La fonction de Rosenbrock (aussi appelée col de Rosenbrock ou $2^{ème}$ fonction de De Jong) est également une fonction à deux variables de $\mathbb{R}^2$ dans $\mathbb{R}$. Pour tout couple $(x,\ y)$, on la calcule avec la formule suivante :

$$\mathcal{F}_2(x,\ y) = 100\ (y - x^2)^2 + (1 - x)^2$$

La fonction est représentée sur la figure 3.5, ainsi que les 25 points utilisés pour effectuer l'interpolation. Les 4 même méthodes ont été implémentées et testées sur ce problème et les résultats sont disponibles dans la table 3.3 avec le nombre de paramètres à estimer pour chaque méthode, l'erreur quadratique moyenne normalisée ou non (nMSE et MSE) et le coefficient de corrélation de Pearson. Les interpolations résultantes sont représentées sur la figure 3.6. La figure 3.7 indique l'amélioration attendue prédite par le modèle de *kriging*.

Là encore, les différents critères comparatifs entre interpolation et fonction à interpoler sont calculés en échantillonnant les fonctions sur une grille régulière de $201 \times 201 = 40401$ points équirépartis dans l'espace d'entrée $[-2,\ 2]^2$. Les valeurs de la fonction sont dans l'intervalle $[0,\ 3609]$.

Ce second test sur la fonction de Rosenbrock ne permet pas de distinguer les résultats des techniques d'interpolation par RBF, par *kriging* ordinaire et par IDW. Ces trois méthodes obtiennent des erreurs moyennes faibles ($0.20 < \text{nMSE} < 0.28$), ainsi que des valeurs de coefficients de corrélation supérieures à 0.95.

Sur ce second exemple avec une fonction plus complexe, la méthode d'interpolation par *IDW* est clairement meilleure : elle obtient des résultats aussi bons que les méthodes par RBF et par *kriging* ordinaire, sans introduire de paramètres à estimer.

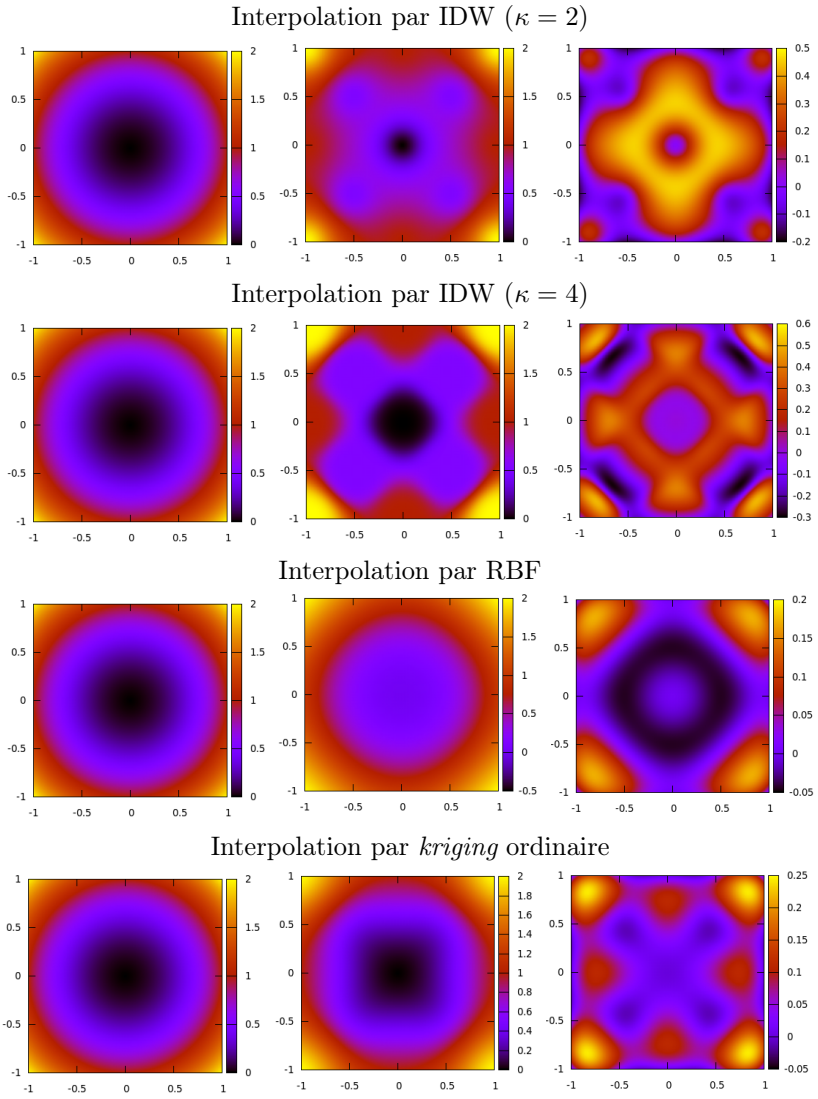

FIGURE 3.3 – Résultats obtenus avec chaque méthode d'interpolation pour la fonction de De Jong. Chaque ligne correspond à une méthode et indique la fonction initiale à gauche, la fonction interpolée au centre et l'erreur commise à droite.

Modèle	Paramètres	MSE	nMSE	Corrélation
Inverse Distance Weighting (κ=2)	0	65541	0.211	0.972
Inverse Distance Weighting (κ=4)	0	77225	0.270	0.965
Radial Basis Functions	25	74973	0.272	0.976
Ordinary kriging	25 + 4	50851	0.197	0.982

TABLE 3.3 – Erreurs et corrélations entre la fonction initiale et l'approximation obtenue avec chaque type de modèle indiqué pour la fonction de Rosenbrock. Le nombre de paramètres à estimer est aussi indiqué pour chaque méthode.

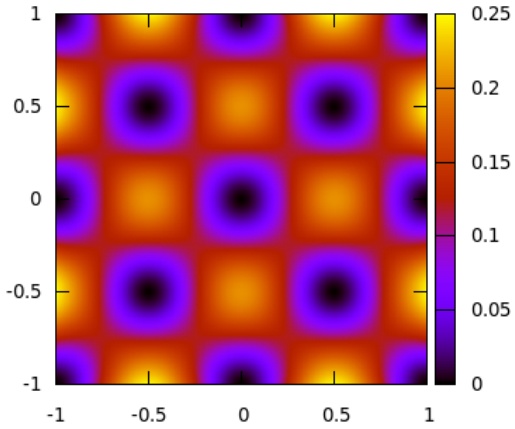

FIGURE 3.4 – Estimation de l'amélioration attendue prédite par le modèle de *krigeage* obtenu pour la fonction de De Jong. Une valeur élevée correspond à une zone potentiellement intéressante pour améliorer l'interpolation.

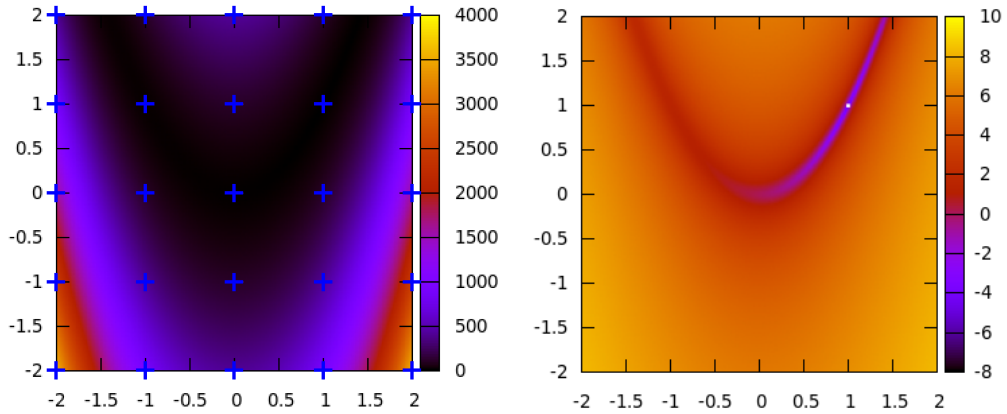

FIGURE 3.5 – Représentation de la fonction de Rosenbrock avec $(x,\ y) \in [-2,\ 2]^2$ – Les croix bleues correspondent aux 25 points utilisés pour l'interpolation – Echelle logarithmique à droite.

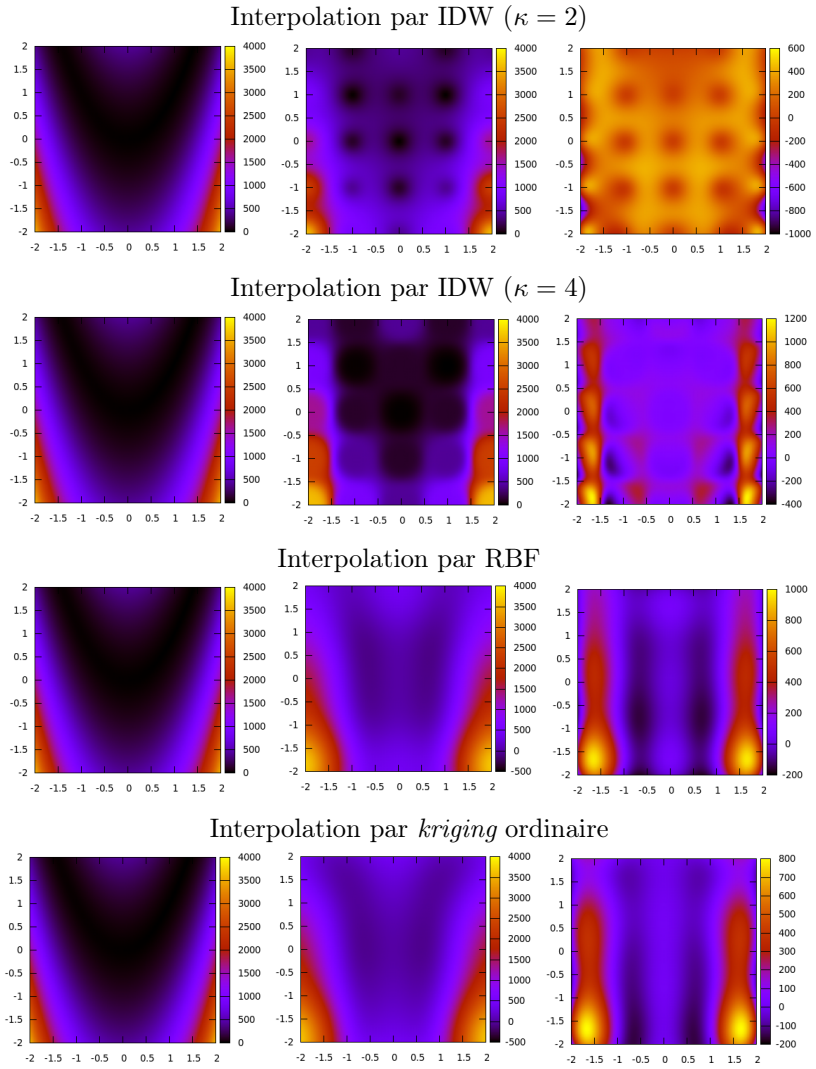

FIGURE 3.6 – Résultats obtenus avec chaque méthode d'interpolation pour la fonction de Rosenbrock. Chaque ligne correspond à une méthode et indique la fonction initiale à gauche, la fonction interpolée au centre et l'erreur commise à droite.

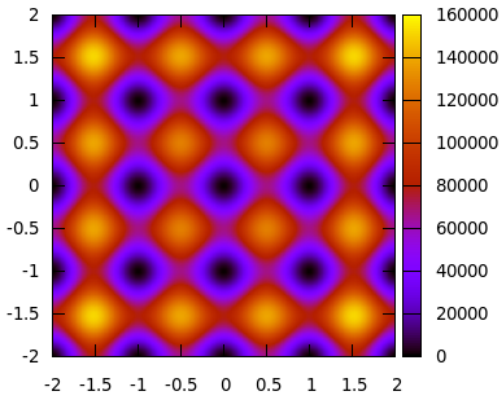

FIGURE 3.7 – Estimation de l'amélioration attendue prédite par le modèle de *kriging* obtenu pour la fonction de De Jong. Une valeur élevée correspond à une zone potentiellement intéressante pour améliorer l'interpolation.

3.2.3 Conclusions

Ce comparatif entre ces différentes méthodes classiques d'interpolation permet surtout de visualiser les caractéristiques de chaque technique sur des problèmes simplistes d'approximation. On remarquera principalement que les interpolations par IDW, RBF et *kriging* ont des performances plutôt proches, aussi bien en terme d'erreur, qu'en terme de corrélation à la fonction cible. Par contre, les modèles de RBF et de *kriging* introduisent sensiblement plus de paramètres dépendant du nombre de points utilisés pour construire l'interpolation et, pour le modèle de *kriging* seulement, du nombre de variables d'entrée de la fonction à interpoler.

3.3 Utiliser un modèle de subsitution pour optimiser

3.3.1 Modèle global ou itératif

L'utilisation d'un modèle de substitution dans le cadre d'un processus d'optimisation n'est pas rare. Le but n'est plus seulement d'obtenir une approximation de la surface à interpoler, mais également de trouver les extrema de cette surface. Indépendamment de la structure utilisée, les différentes méthodes permettant de générer des modèles pour un besoin d'optimisation se répartissent en deux classes suivant si elles sont basées sur un modèle global [Simpson *et al.* 2001, Queipo *et al.* 2005] ou un modèle itératif [Sasena 2002, Jin 2005, Hemker *et al.* 2009].

Dans le premier cas, on se trouve dans un contexte d'optimisation classique : le modèle de substitution devient la fonction de fitness à optimiser et on cherche simplement la solution qui maximise la valeur de performance prédite par le modèle. Le processus peut se découper en trois étapes :

1. Evaluation de la fonction en n points de l'espace de recherche.
2. Interpolation du modèle à partir de ces n points.

3. Recherche de l'optimum prédit par le modèle.

Néanmoins, comme l'échantillonnage initial des solutions est souvent insuffisant pour avoir une bonne vision des zones d'intérêt de l'espace de recherche, le modèle est souvent mis à jour en évaluant quelques solutions au cours de l'optimisation suivant une stratégie de mise à jour [Jin 2005] :

1. Evaluation exacte de n_{init} points de l'espace de recherche.
2. Tant que le modèle n'est pas assez bon ou qu'un point suffisamment performant n'a pas été trouvé :
   a. interpolation du modèle à partir des points connus ;
   b. évaluer un nouveau point d'après une stratégie donnée.
3. Recherche de l'optimum à l'aide du modèle (simultanément à l'étape 2).

L'utilisation d'un modèle itératif introduit un schéma d'optimisation plus compliqué, car le modèle est modifié au cours du processus : la fonction de fitness est donc dynamique.

La question principale pour choisir entre ces deux techniques est la suivante : est-il possible d'échantillonner correctement l'espace de recherche avant l'optimisation ? Par exemple, imaginons que nous cherchions à trouver le minimum global sur la fonction de Rosenbrock (cf. figure 3.5), il est plus intéressant de se concentrer sur la région de la vallée que d'échantillonner de manière équilibrée sur tout l'espace. Or, tant qu'un modèle approché de la fonction n'est pas construit, il est difficile de biaiser le choix de nouveaux points vers des zones d'intérêt, ici vers les zones où la fitness est basse. Dans le cas où peu d'information est disponible sur les caractéristiques de la fonction à optimiser, utiliser un modèle itératif permet d'éviter de contraindre les résultats par un échantillonnage arbitraire.

3.3.2 Les différentes types de stratégie

Un échantillonnage *a priori* peut ne pas permettre d'avoir une vue représentative des zones d'intérêt de la fonction et, dans ce cas, construire un modèle global ne permettra pas forcément trouver un bon optimum [Sasena 2002]. Un second problème peut se poser, même si le modèle global est une bonne approximation de l'ensemble de la fonction : l'interpolation peut introduire de faux optima globaux. Par exemple, si on considère les résultats obtenus avec la méthode de *kriging* ordinaire sur la fonction de Rosenbrock, on s'aperçoit que l'interpolation introduit un minimum global négatif au point $(0.6,\ 0.12)$, bien que l'approximation soit globalement de bonne qualité. Ainsi une optimisation effectuée directement sur ce modèle global ne trouverait pas le véritable minimum de valeur nulle au point $(1,\ 1)$.

Une alternative à un échantillonnage *a priori* consiste à mettre à jour le modèle pendant l'optimisation en sélectionnant de manière active les prochains points à évaluer. On peut distinguer deux techniques principales [Jones 2001]. La première revient à systématiquement évaluer le point de performance maximale sur l'approximation [Moraglio & Kattan 2011, Koziel & Bandler 2007] : ceci permet de guider l'optimisation dans des zones de hautes valeurs de fitness. On arrête le processus une fois qu'une solution satisfaisante a été trouvée ou encore quand la fitness des

solutions évaluées n'augmente plus suffisamment. Une méthode similaire peut être dérivée dans le cas de l'optimisation multi-objectif [Voutchkov & Keane 2010] : on ne transfère plus le point de performance maximale, mais un ou plusieurs points non-dominés. Le problème habituel de ce type de stratégie provient du fait qu'elle guide rapidement l'optimisation vers des optima locaux, car n'exploitant que la surface d'interpolation sans chercher à explorer d'autres zones de l'espace de recherche potentiellement plus intéressantes [Jones *et al.* 1998]. Le résultat sera aussi très dépendant de l'initialisation du modèle de substitution.

La seconde stratégie habituellement utilisée revient à sélectionner des points dans les zones encore mal modélisées de l'espace, c'est-à-dire dans les zones où on connaît peu de points ou les zones les plus éloignées des points connus actuels. On cherche dans ce cas à explorer l'espace de recherche pour avoir une vue d'ensemble correcte de la fonction à interpoler. Si l'interpolation se fait à l'aide d'un modèle de *kriging*, il est possible d'utiliser la fonction d'"amélioration attendue pour évaluer les nouveaux points [Leary *et al.* 2004, Huang *et al.* 2006] : pour tous les nouveaux points potentiels considérés, on calcule la valeur d'amélioration attendue et on évalue le nouveau point qui maximise cette valeur, donc qui apporte apparemment le plus d'information à l'approximation. Cette fonction d'amélioration permet également de stopper l'optimisation lorsque l'amélioration attendue devient trop faible. Le désavantage de cette stratégie est qu'elle nécessite potentiellement beaucoup d'évaluations avant de converger vers l'optimum global [Jones 2001].

De récentes tentatives visent à coupler ces deux types de stratégie. L'approche hybride la plus simple revient à alterner les étapes d'exploration et d'exploitation suivant une fréquence fixe [Sasena 2002] en se basant respectivement sur l'amélioration attendue et sur la performance prédite. Il est également possible d'introduire une hiérarchie entre les critères : l'optimum de la surface d'interpolation est évalué s'il est suffisamment éloigné de points déjà évalués, sinon on évalue le point qui maximise l'amélioration attendue [Hemker *et al.* 2006, Hemker *et al.* 2009]. Afin de gérer les deux aspects indépendamment, les solutions peuvent également être évaluées par deux objectifs distincts dans le cadre d'un processus d'optimisation multi-objectif [Schaul *et al.* 2011] : la performance d'après l'interpolation et un terme de gain d'information similaire à l'amélioration attendue. Le prochain point à évaluer est alors choisi aléatoirement parmi les solutions non-dominées d'après ces deux objectifs.

Il est difficile d'évaluer l'apport des stratégies hybrides par rapport aux stratégies plus classiques, étant donné le peu de travaux qui les comparent. Le point à retenir est que la qualité finale de l'optimum trouvé dépend tout autant de la stratégie de mise à jour que de la structure du modèle de substitution.

3.4 Conclusion

Dans ce chapitre, nous avons vu que les modèles de substitution pouvaient être utilisés dans un cadre d'optimisation afin de réduire le coût expérimental lié à l'éva-

luation des solutions. Afin de réduire au maximum le nombre d'évaluations exactes effectuées, on va donc préférer une structure de modèle et une stratégie de mise à jour qui vérifient les deux points suivants.

- L'initialisation du modèle de substitution nécessite peu d'expériences, i.e. la structure choisie doit introduire peu de paramètres.
- La stratégie de mise à jour assure un bon compromis entre exploitation et exploration, i.e. elle mène aux zones optimales en un nombre restreint d'évaluations.

Basé sur les résultats obtenus avec deux fonctions de tests, les deux méthodes qui semblent faire de bons compromis entre la fiabilité, la simplicité d'utilisation et le coût expérimental sont l'interpolation par IDW, l'interpolation par RBF et les méthodes de *kriging*. Le choix final dépend principalement du coût expérimental maximal que l'utilisateur s'autorise pour l'initialisation des modèles, ainsi que de la taille de l'espace d'entrée du modèle de substitution pour l'interpolation par *kriging*.

Chapitre 4

Approche par transférabilité et passage à la réalité

Le contenu de ce chapitre a fait l'objet d'une publication à la conférence *GECCO* en 2010 dans l'article [Koos *et al.* 2010], ainsi que d'un article de journal dans la revue *IEEE Transactions on Evolutionary Computation* [Koos *et al.* 2011].

4.1 L'approche par transférabilité

4.1.1 Principes

Le problème du passage de la simulation à la réalité ou *reality gap* [Jakobi *et al.* 1995] est un problème clé en robotique évolutionniste et, plus généralement, dès qu'on veut optimiser ou apprendre un comportement en simulation avant de le transférer sur le système réel : une bonne performance en réalité n'est pas nécessairement observée [Brooks 1992, Jakobi *et al.* 1995]. Pour comprendre l'importance de ce problème, il faut d'abord constater qu'optimiser directement sur le robot est souvent difficile en pratique et va imposer des contraintes drastiques sur le nombre d'évaluations et donc sur la qualité de la solution finale trouvée [Floreano & Mondada 1998, Hornby *et al.* 2002, Kohl & Stone 2004, Zykov *et al.* 2004, Regan *et al.* 2006, Gongora *et al.* 2009, Yosinski *et al.* 2011]. Le recours à une simulation est alors obligatoire en tant que moyen d'évaluer les différentes solutions dans un processus sécurisé et rapide [Harvey *et al.* 1992]. Une méthode classique revient à optimiser les contrôleurs directement en simulation et de transférer le plus performant sur le robot : si le simulateur est suffisamment bon, le comportement observé en réalité devrait correspondre au comportement simulé. Cependant, des simulations précises peuvent parfois être plus lentes qu'une évaluation directe sur le robot : pour obtenir des modèles de simulation compatibles avec un processus d'optimisation itératif, il est parfois nécessaire de négliger certains phénomènes physiques dont la prise en compte augmente sensiblement la durée d'évaluation des solutions. Ces simulateurs simplifiés sont certes moins précis, mais également plus rapides. Certaines applications ne permettent même pas de construire de simulateurs précis : par exemple la locomotion de certains robots mobiles peut impliquer des interactions roues-sol ou pattes-sol complexes et les modèles de simulation utilisés sont souvent basés sur des hypothèses simplificatrices [Weingarten *et al.* 2004, Lhomme-Desages *et al.* 2007, Terekhov *et al.* 2010a].

Si le processus d'optimisation s'effectue uniquement en simulation, l'algorithme

évolutionniste peut possiblement exploiter les parties peu réalistes de la simulation pour obtenir des performances apparemment hautes : l'optimisation est opportuniste, parfois au détriment de de la performance finale sur le robot des solutions trouvées. A partir du moment où la fiabilité des différentes parties du modèle n'est pas prise en compte pendant l'optimisation, l'algorithme va seulement chercher à maximiser la fonction de fitness en simulation. Il est d'ailleurs difficile de déterminer a priori quelles parties d'une simulation dynamique seront fiables ou non. Si le comportement final trouvé exploite des parties du simulateur non réalistes, il est très probable que le comportement correspondant en réalité sera sensiblement différent et souvent significativement moins performant. En pratique, même si beaucoup de travaux de robotique évolutionniste semblent donner des résultats très prometteurs avec des comportements robotiques originaux et complexes en simulation [Prieto *et al.* 2010, Terekhov *et al.* 2010b], peu de travaux en comparaison présentent des résultats effectifs sur des plates-formes robotiques, vraisemblablement en partie à cause de problèmes de transfert. En résumé, le problème du passage de la simulation à la réalité rend peu fiable un processus d'optimisation uniquement basé sur un modèle de simulation. Nous sommes donc confrontés à deux aspects antagonistes : (1) optimiser sur le robot est long, coûteux, parfois risqué ; (2) optimiser en simulation ne permet pas toujours d'obtenir une solution performante sur le robot.

Ce problème n'est en fait pas uniquement une question spécifique à la robotique évolutionniste, mais touche potentiellement tout travail qui vise à optimiser un processus sans pouvoir l'évaluer dans l'environnement ciblé. L'optimisation se fera alors dans un environnement simplifié (ici une simulation), mimant les caractéristiques de l'environnement ciblé. Si le comportement de la solution optimale est basé sur des parties de la simulation qui diffèrent trop des phénomènes réels modélisés, on observera nécessairement un comportement différent et souvent moins performant dans l'environnement ciblé. Ce problème est par exemple illustré dans le cas d'optimisation (sans processus évolutionniste) de structures de contrôle pour un quadri-rotor [Adigbli *et al.* 2007], malgré l'utilisation d'un modèle des dynamiques de quadri-rotor relativement perfectionné comme environnement d'évaluation. Il peut même y avoir des problèmes similaires à ceux d'un passage de la simulation à la réalité, et ce même quand l'optimisation se fait directement sur le robot mais dans un contexte simplifié. Le cas a notamment été observé sur l'optimisation de contrôleurs pour un petit hélicoptère [Gongora *et al.* 2009] : afin d'éviter des comportements dangereux sur le robot, l'évaluation pendant l'optimisation se faisait en accrochant l'appareil à un portique, menant à des comportements différents quand l'hélicoptère est testé librement.

Comme nous l'avons montré en présentant différents travaux de l'état de l'art (voir partie 2.4.1), le problème du passage de la simulation à la réalité n'a pas encore été résolu de manière générale. A part l'approche de Jakobi proposant l'évolution de contrôleurs dans une ou plusieurs simulations minimales [Jakobi 1998a], il n'existe pas de méthode formalisée applicable dans le cas général. Le but de cette partie est d'introduire une méthodologie générale, l'approche par transférabilité, qui permet de résoudre ce problème de passage de la simulation à la réalité et de réconcilier la

robotique évolutionniste avec l'usage de simulateurs.

Cette approche part du principe que certaines parties du simulateur sont suffisamment réalistes pour modéliser correctement certains comportements. En effet, bien que les modèles de simulation utilisés ne soient pas parfaits, ils sont construits afin de modéliser précisément un ensemble de phénomènes physiques. Les comportements performants en simulation qui vont s'appuyer sur ces phénomènes bien modélisés devraient bien se transférer à la réalité et donc obtenir de bonnes performances sur le robot. L'approche par transférabilité vise à optimiser les contrôleurs à l'aide d'un simulateur, tout en se focalisant sur les parties de ce simulateur qui vont être réalistes pour assurer de bons transferts. On cherche alors à :

- trouver des contrôleurs à la fois performants sur la tâche considérée en simulation et se transférant bien sur le robot physique ;
- effectuer aussi peu d'expériences sur le robot pendant l'optimisation pour déterminer quelles parties de la simulation sont réalistes ou non.

Pour reprendre la classification introduite dans l'état de l'art (voir partie 2.4.1), l'approche par transférabilité fait partie des méthodes d'optimisation en simulation à l'aide du robot. Contrairement aux approches qui cherchent à améliorer le simulation utilisé [Bongard & Lipson 2005, Bongard *et al.* 2006, Zagal *et al.* 2008, Koos *et al.* 2009, Zagal *et al.* 2009], le simulateur reste fixe pendant tout le processus. Notre première hypothèse, illustrée sur la figure 4.1 regroupe deux idées.

- Certaines parties du simulateur sont suffisamment fiables pour assurer de bons transferts sur le robot.
- La fonction de fitness en simulation ne fournit pas un bon gradient à l'optimisation : les solutions les plus performantes en simulation ne sont pas nécessairement transférables sur le robot physique et génèrent des comportements peu performants.

L'approche par transférabilité a pour but de trouver des solutions efficaces qui exploitent plutôt les phénomènes bien modélisés par le simulateur, donc là où l'évaluation en simulation sera proche de l'évaluation sur le robot physique. La qualité du transfert d'un contrôleur donné sur le robot est notée par une mesure de transférabilité qui compare le comportement en simulation et le comportement en réalité correspondant au contrôleur considéré : plus les comportements sont proches, plus la valeur de transférabilité sera élevée. Afin de prendre en compte ce nouveau objectif, l'approche par transférabilité reformule le problème d'optimisation en un processus multi-objectif qui vise à trouver des solutions faisant compromis entre : (1) la performance sur la tâche en simulation ; (2) l'objectif de transférabilité.

Le calcul de la valeur de transférabilité nécessite d'avoir à disposition le comportement en réalité correspondant à la solution considérée. Il est donc évident que nous ne pourrons pas calculer la valeur exacte de transférabilité pour chaque individu, ce qui imposerait autant d'expériences qu'une optimisation directe sur le robot. Notre seconde hypothèse repose sur le fait que, connaissant les valeurs exactes de transférabilité pour quelques individus, c'est-à-dire en transférant quelques individus sur le robot, il est possible construire une approximation de la fonction de transférabilité, par interpolation par exemple, et donc en dériver des valeurs approchées pour tous les

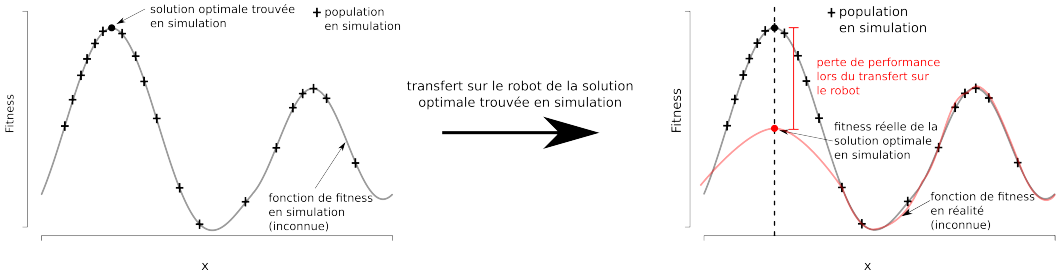

FIGURE 4.1 – Illustration du problème du passage de la simulation à la réalité sur un problème fictif d'optimisation à 1 dimension (on cherche à maximiser la fitness). Certaines parties du simulateur sont suffisamment fiables pour assurer de bons transferts sur le robot. Maximiser la fonction de fitness en simulation mène cependant à des solutions non transférables en réalité : une importante perte de performance est observée lors du transfert. En se limitant aux parties fiables du simulateur, on aurait pu obtenir une meilleure solution en réalité.

contrôleurs non transférés. Optimisée conjointement avec l'objectif de performance, cette transférabilité approchée permettrait alors de guider le processus évolutionniste vers des solutions-compromis, à la fois efficaces en simulation et transférables sur le robot physique, le tout en assurant un coût réduit en temps expérimental. Le processus complet est représenté sur la figure 4.2.

Dans ce travail, la transférabilité d'un contrôleur est calculée par une mesure de disparité entre simulation et réalité notée disparité $S|R$. Cette mesure estime les variations entre le comportement simulé et le comportement réel correspondant au contrôleur considéré : plus la valeur de disparité $S|R$ est élevée, moins la transférabilité est bonne. Pour les raisons évoquées précédemment, la disparité $S|R$ de chaque solution ne peut pas être évaluée exactement dans un temps raisonnable. Cette mesure de disparité est alors approximée en utilisant un modèle de substitution construit par interpolation à l'aide de quelques expériences effectuées sur le robot pendant l'optimisation. Le choix des contrôleurs à transférer est assuré par une stratégie de mise à jour définie en fonction de la structure du modèle de substitution utilisée.

Comme l'approche par transférabilité a pour but de trouver des solutions à la fois efficaces et transférables, elle ne va pas toujours trouver la solution optimale en réalité, mais de bons compromis entre performance et transférabilité. Par exemple, si la solution optimale en réalité met en jeu un comportement mal modélisé en simulation, elle sera perçue comme non transférable en simulation et donc rejetée. Ce problème est illustré sur la figure 4.3. L'utilisation de l'approche par transférabilité repose sur l'hypothèse que les parties réalistes de la simulation incluent des comportements suffisamment performants par rapport à la tâche à résoudre. Nous supposons que ce cas est toujours vérifié pour des applications réalistes en robotique, car les modèles mécaniques utilisés comme simulateurs sont construits pour

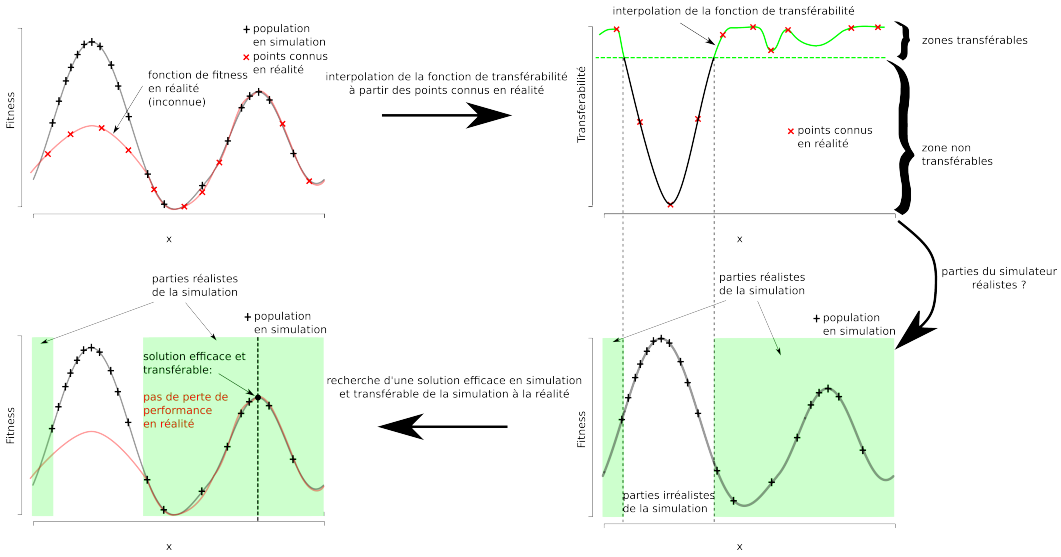

FIGURE 4.2 – Illustration de l'approche par transférabilité. Si certains points sont connus en réalité, il est possible d'interpoler une approximation de la fonction de transférabilité. Cette approximation permet alors de distinguer les parties bien modélisées en simulation des zones insuffisamment réalistes : on focalise alors la recherche sur les solutions efficaces en simulation et transférables du simulateur à la réalité.

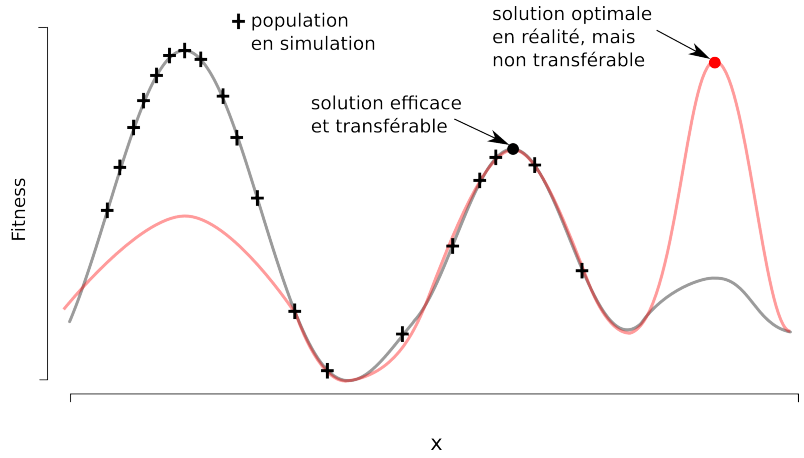

FIGURE 4.3 – La solution optimale en réalité peut correspondre à un comportement non-transférable en simulation. Dans ce cas, comme l'approche par transférabilité cherche des solutions à la fois performantes et transférables, la solution optimale en réalité ne sera vraisemblablement pas sélectionnée. Nous supposons que ce cas de figure n'arrive pas pour des applications réalistes en robotique, car les modèles mécaniques utilisés comme simulateurs sont construits pour modéliser les phénomènes les plus importants intervenant dans la tâche considérée.

modéliser les phénomènes les plus importants intervenant dans la tâche considérée.

Un autre cas peut théoriquement poser problème. Comme illustré sur la figure 4.4, la solution optimale en simulation peut aussi l'être en réalité, tout en étant non transférable à cause d'une perte de performance significative lors du transfert. Comme l'approche par transférabilité dirige la recherche dans les zones transférables, cette solution optimale n'est pas sélectionnée. Quoiqu'il en soit, ce cas peut être facilement détecté : la solution trouvée par l'approche par transférabilité aura systématiquement une fitness sur le robot inférieure à la fitness réelle de la solution la plus performante trouvée en simulation. En se basant sur cette observation, ce cas de figure n'a pas été observé dans les deux expériences présentées dans cette partie.

4.1.2 De la disparité $S|R$ exacte à un modèle de substitution

On appelle D^* la fonction de disparité entre simulation et réalité (disparité $S|R$) : pour tout contrôleur c, la valeur exacte $D^*(c)$ de disparité $S|R$. La valeur exacte de disparité $S|R$ est calculée en comparant le comportement simulé et le comportement réel correspondant à l'individu c considéré. Une telle fonction de disparité $S|R$ ne peut donc pas être directement utilisée comme objectif à optimiser, car chaque solution de la population devrait être transférée sur le robot afin d'obtenir son comportement en réalité.

Pour limiter drastiquement le nombre d'expériences menées sur le robot, l'objectif à optimiser ne sera pas directement la valeur exacte de disparité $S|R$ mais une valeur approchée basée sur un modèle de substitution de cette fonction. L'usage

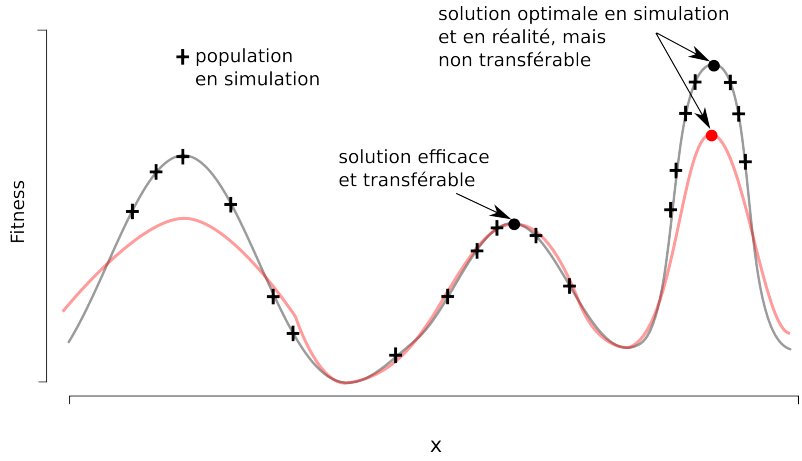

FIGURE 4.4 – La solution optimale en simulation peut aussi être optimale en réalité malgré une différence de performance significative lors du transfert sur le robot. L'approche par transférabilité devrait donc éviter cette zone optimale en simulation et ne pas sélectionner la solution optimale en réalité. Quoi qu'il en soit, ce cas peut être facilement détecté, car la fitness réelle de la meilleure solution trouvée avec l'approche par transférabilité devrait être systématiquement plus basse que la fitness réelle de la solution la plus performante trouvée en simulation. Ce cas n'a jamais été observé dans les deux expériences présentées dans ce chapitre.

des modèles de substitution dans des problèmes d'optimisation intervient habituellement quand évaluer une solution potentielle implique de longues expériences et/ou un coût expérimental important [Jones 2001, Jin 2005, Voutchkov & Keane 2006, Hemker *et al.* 2009]. Ainsi, au lieu d'utiliser directement le système physique considéré comme environnement d'évaluation, les solutions sont optimisées à l'aide d'un modèle approché de ce système. L'utilisation d'un modèle de substitution implique trois aspects :

- le choix d'un espace d'entrée sur lequel construire l'approximation ;
- le choix d'une structure de modèle ;
- le choix d'une stratégie de mise à jour.

La construction d'un modèle de substitution de la fonction exacte de disparité D^* se fait sur un espace d'entrée donné, i.e. un ensemble de variables utilisées pour la prédiction de la valeur approchée. Cet espace d'entrée peut être ici compris comme la description d'un individu : à l'aide de sa description, le modèle déterminera si le contrôleur évalué est plutôt transférable ou non. Dans le cadre d'une expérience de robotique évolutionniste, nous avons deux choix principaux : le génotype ou une description comportementale générée via la simulation. En nous basant sur différents travaux indiquant l'intérêt des descriptions comportementales pour des applications robotiques, nous avons fait ici le choix d'utiliser un comportement en entrée du modèle de substitution dans la définition de l'approche par transférabilité (voir partie 2.3.2 pour une comparaison plus détaillée entre description génotypique et description comportementale). Ce point sera étudié plus en détail dans les sec-

tions 4.3.4 et 4.4.3 en comparant l'approche par transférabilité à une méthode où le modèle de substitution est défini à partir du génotypique.

Dans la section II traitant en détail de l'utilisation des modèles de substitution, nous avons présenté différents types de structures de modèle basés sur des techniques d'interpolation, ainsi que différentes stratégies de mise à jour. Concernant le choix de la structure, nous avons choisi une des méthodes les plus simples, l'interpolation par Inverse Distance Weighting [Shepard 1968] (IDW). La structure du modèle par IDW dépend d'un unique paramètre κ définissant la régularité de la surface interpolée. Etrangement, cette méthode est très rarement utilisée dans un cadre d'optimisation et est souvent absente des travaux utilisant les algorithmes évolutionnistes au profit de techniques basées sur l'interpolation par Kriging [Jones 2001, Jin 2005]. Or, l'interpolation par IDW présente plusieurs avantages :

- Elle n'introduit aucun paramètre à estimer et ne nécessite donc pas beaucoup d'expériences pour initialiser le modèle (une seule expérience suffit).
- Malgré sa simplicité, elle est compétitive avec les méthodes de Kriging [Malvić & Durekovic 2003, Mueller *et al.* 2004] dans le cas où peu d'informations sont disponibles sur la corrélation spatiale des variables, ce qui est le cas ici.
- Le modèle est très rapide à construire et à utiliser comme prédicteur.

Un autre point important à souligner concerne les méthodes de Kriging : le nombre de paramètres à estimer pour cette méthode dépend du nombre de variables d'entrée de la fonction à approximer, c'est-à-dire ici du nombre de descripteurs comportementaux utilisés pour décrire un individu en simulation. Si l'espace d'entrée compte peu de paramètres, l'interpolation par Kriging sera envisageable. Mais dans le cas de descriptions comportementales denses (trajectoire du robot, valeurs des capteurs au cours du temps, ...), recourir à une technique de Kriging imposera d'effectuer un nombre d'expériences élevé, et ce seulement pour initialiser le modèle. Afin de valider le choix de l'interpolation par IDW, une comparaison entre l'approche par transférabilité et une méthode basée sur l'interpolation par Kriging sera envisagée dans la partie 4.4.3.

Concernant le choix de la stratégie de mise à jour du modèle, ce point est détaillé dans la partie 4.2.3.

4.1.3 Métrique *in silico* b_{dist}

Pour utiliser l'interpolation par IDW, une fonction de distance entre les différents individus doit être définie sur les variables de l'espace d'entrée du modèle de substitution. Cette fonction de distance appelée métrique *in silico* va donc ici être basée sur une description comportementale. Pour chaque contrôleur évalué en simulation, le comportement correspondant est décrit par e valeurs réelles que nous appellerons caractéristiques comportementales. Une fois connues, ces caractéristiques permettent de définir une distance comportementale simple entre individus en simulation, la métrique *in silico*. Etant donnés les vecteurs de caractéristiques comportementales en simulation $\mathbf{b^{(1)}}$ et $\mathbf{b^{(2)}}$ correspondant aux contrôleurs $c^{(1)}$ et $c^{(2)}$, nous utilisons

comme métrique *in silico* b_{dist} entre ces deux contrôleurs la distance euclidienne entre $\mathbf{b^{(1)}}$ et $\mathbf{b^{(2)}}$:

$$b_{dist}(c^{(1)}, c^{(2)}) = ||\mathbf{b^{(1)}} - \mathbf{b^{(2)}}||$$

Cette métrique *in silico* a l'avantage de permettre de comparer les contrôleurs rapidement en simulation sans dépendance directe à leurs génotypes ou phénotypes.

4.1.4 Modèle de substitution de la disparité $S|R$

Si des contrôleurs ont déjà été transférés sur le robot (au moins 1) et que les valeurs exactes de disparité $S|R$ correspondantes ont été calculées, un modèle de substitution $\hat{D}$ de la fonction D^* de disparité $S|R$ peut être interpolé par IDW. Soit $\mathcal{C}$ l'ensemble de tous les contrôleurs et $\mathcal{C}_T \subset \mathcal{C}$ l'ensemble des contrôleurs déjà transférés en réalité ($|\mathcal{C}_T| > 1$), soit $D^*(c_i)$ la disparité $S|R$ exacte correspondant à chaque contrôleur $c_i \in \mathcal{C}_T$, le modèle de substitution $\hat{D}$ de la disparité $S|R$ est construit comme suit :

$$\forall c \in \mathcal{C}, \hat{D}(c) = \frac{\sum_{c_i \in \mathcal{C}_T} D^*(c_i)\ b_{dist}(c_i, c)^{-\kappa}}{\sum_{c_i \in \mathcal{C}_T} b_{dist}(c_i, c)^{-\kappa}}.$$

Dans les deux applications considérées dans ce chapitre, nous avons choisi arbitrairement de fixer κ à 2, permettant d'obtenir une surface interpolée relativement régulière (pour plus de détails, se reporter à la section 3.1.3).

4.2 Schéma d'optimisation

4.2.1 Objectifs d'évaluation

Tout contrôleur de la population est évalué par les trois objectifs suivants :

1. performance sur la tâche en simulation ;
2. valeur approchée de disparité $S|R$ correspondante obtenue à l'aide du modèle de substitution ;
3. objectif de diversité comportementale.

Ce troisième objectif est introduit afin de maintenir une diversité comportementale au sein de la population, ce qui permet d'explorer plus efficacement l'espace de recherche [Mouret & Doncieux 2009b, Doncieux & Mouret 2010, Mouret & Doncieux 2011]. Plus de détails sur la nécessité d'utiliser cet objectif sont disponibles dans la section 2.3.3.

4.2.2 Diversité comportementale

Afin de quantifier la diversité d'un contrôleur donné par rapport aux contrôleurs qui ont déjà été transférés en réalité, nous définissons l'objectif de diversité comportementale comme suit. Soit $\mathcal{C}_T$ l'ensemble des contrôleurs déjà transférés en

réalité et b_{dist} la métrique *in silico*, la valeur de diversité comportementale pour un contrôleur c donné est :

$$div(c) = \min_{c_i \in \mathcal{C}_T} \; b_{dist}(c,\; c_i)$$

Cette valeur de diversité ne dépend pas directement du génotype et du phénotype de la solution concernée, mais uniquement de ses caractéristiques comportementales, et favorise les solutions dont les comportements sont les plus différents de ceux des contrôleurs déjà transférés.

4.2.3 Stratégie de mise à jour du modèle de substitution

L'utilisation d'un modèle de substitution implique généralement le choix d'une stratégie de mise à jour, notamment quand un échantillonnage *a priori* de l'espace de recherche ne permet pas de capter les zones d'intérêt. Cette stratégie définit une heuristique utilisée pour sélectionner les expériences qui seront effectuées sur le dispositif physique afin d'améliorer le modèle de manière pertinente. Comme le modèle d'IDW ne permet pas de dériver une fonction d'amélioration attendue contrairement aux méthodes de Kriging (voir partie 3.1.5), nous avons choisi de définir une stratégie simple qui sélectionne les contrôleurs dont la valeur de diversité comportementale par rapport aux contrôleurs déjà transférés est supérieure à un seuil fixé τ_{div}. Ceci évite que les expériences effectuées soient trop proches les unes des autres dans l'espace comportemental, risquant d'apporter peu d'informations au modèle.

A chaque génération, un contrôleur est aléatoirement sélectionné parmi ceux dont la diversité comportementale est supérieure à τ_{div} et est transféré sur le robot. Si aucun contrôleur de la population n'a une valeur de diversité suffisante, aucun transfert ne sera effectué pendant la génération concernée. Le seuil τ_{div} va donc directement influer sur le nombre moyen d'expériences de transferts effectuées pendant l'optimisation et peut être réglé par l'utilisateur vu qu'il dépend essentiellement du modèle de simulation utilisé. Quand un contrôleur a été transféré, il est ajouté à l'ensemble des contrôleurs déjà transférés $\mathcal{C}_T$ et la valeur de disparité $S|R$ exacte correspondante est utilisée pour mettre à jour le modèle de subsitution.

4.2.4 Etapes de l'algorithme

Afin d'initialiser le modèle de substitution de la disparité $S|R$, nous supposons qu'un contrôleur c_0 a déjà été transféré sur le robot avant le début de l'optimisation et que la valeur de disparité $S|R$ exacte correspondante ainsi que les caractéristiques comportementales observées en simulation sont disponibles. A chaque génération, l'algorithme suivant est utilisé comme représenté sur la figure 4.5.

A. Evaluation de chaque contrôleur :

A1. calcul des caractéristiques comportementales et de la fitness en simulation ;

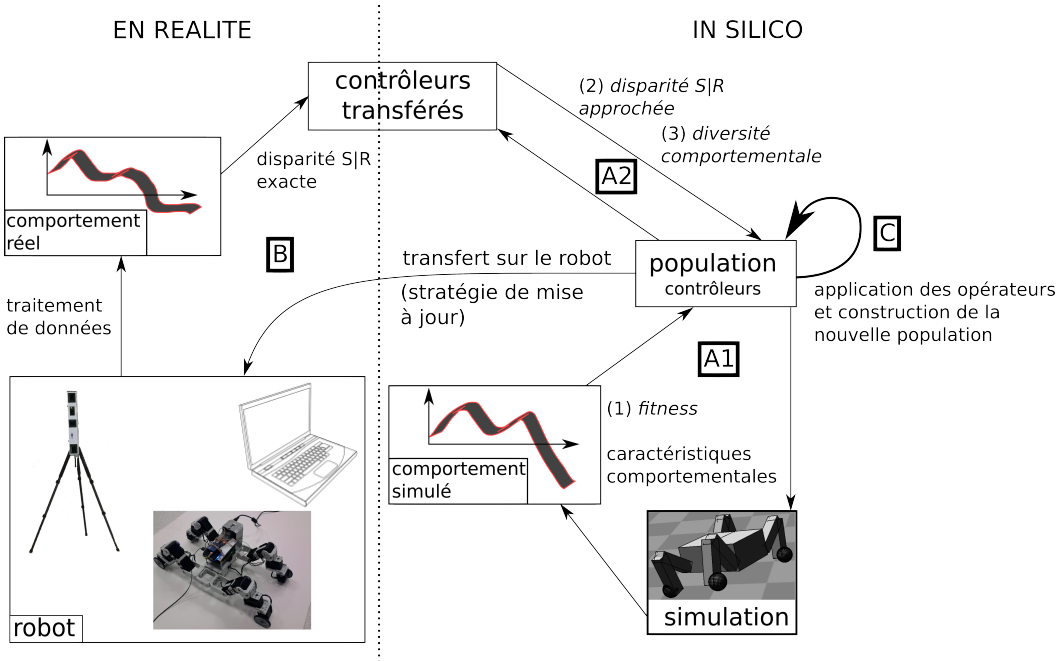

FIGURE 4.5 – Etapes de l'algorithme utilisé à chaque génération dans l'approche par transférabilité – A1. Les caractéristiques comportementales et la fitness sont évaluées en simulation pour chaque contrôleur. A2. Le comportement simulé de chaque contrôleur permet de calculer la valeur approchée de disparité $S|R$ d'après le modèle de substitution construit par IDW, ainsi que la valeur de diversité comportementale. B. Si des individus ont une valeur de diversité suffisamment haute, l'un d'eux est sélectionné et transféré sur le robot physique. Une valeur exacte de disparité $S|R$ est alors calculée pour ce contrôleur en comparant les comportements correspondants observés en simulation et en réalité. C. Les opérateurs évolutionnistes et le mécanisme de sélection permettent de construire la population suivante.

A2. calcul de la disparité $S|R$ approchée et de la valeur de diversité comportementale en fonction des caractéristiques comportementales et de la métrique *in silico*.

B. Si des contrôleurs ont une valeur de diversité suffisamment élevée, l'un d'eux est transféré sur le robot physique.

C. Application des opérateurs évolutionnistes et construction de la population suivante.

4.2.5 Meilleure solution de l'optimisation

Nous supposons tout d'abord qu'un seuil $D^*_{threshold}$ peut être empiriquement défini sur les valeurs de disparité $S|R$ de telle façon que des valeurs de disparité inférieures que $D^*_{threshold}$ correspondent à des transferts de bonne qualité. Il s'agit d'un critère fourni par l'utilisateur qui indique à partir de quelle valeur les comportements en réalité et en simulation sont considérés comme proches.

De manière générale, on peut distinguer deux types d'applications : (A1) l'optimalité d'une solution est connue dès que son comportement est évalué ; (A2) il n'est pas possible de définir un objectif pour distinguer les individus optimaux des individus non optimaux (la valeur maximale de la fitness n'est pas connue). Concernant la classe A1, le but de l'optimisation revient à trouver une solution dont le comportement en réalité vérifie l'objectif d'optimalité, c'est-à-dire un contrôleur qui résout la tâche. Ainsi, comme l'approche par transférabilité impose de transférer quelques contrôleurs pendant l'optimisation, un individu optimal en réalité peut être trouvé avant la fin du processus. Dans ce cas, l'optimisation peut être stoppée et cet individu sera la meilleure solution trouvée.

Si aucun individu optimal en réalité n'a été observé lors des différents transferts pour le type d'application A1, un critère doit être défini afin de sélectionner un "meilleur" compromis parmi les solutions non-dominées. Il en est de même pour le cas A2 où aucune information *a priori* n'est disponible sur l'optimalité éventuelle des solutions. A la fin du processus d'évolution, l'algorithme multi-objectif renvoie l'ensemble des solutions non-dominées en fonction des différents objectifs (ici 3). Nous définissons alors l'ensemble transférable des solutions non-dominées comme le sous-ensemble contenant les solutions non-dominées dont la valeur de disparité $S|R$ est inférieure au seuil $D^*_{threshold}$. Deux cas sont alors envisageables : si cet ensemble est vide, le meilleur compromis est la solution non-dominée avec la valeur minimale de disparité $S|R$, bien qu'elle ne devrait pas se transférer bien. Si l'ensemble n'est pas vide, nous avons à choisir une solution compromis dans cet ensemble transférable.

Construisons tout d'abord le point idéal transférable dont les coordonnées sont les valeurs maximales observées pour chaque objectif dans l'ensemble transférable des solutions non-dominées. Nous sélectionnons alors comme meilleur compromis, la solution dont la distance à ce point idéal approximatif est minimale.

L'ensemble du processus de choix de la meilleure solution est représenté sur la figure 4.6.

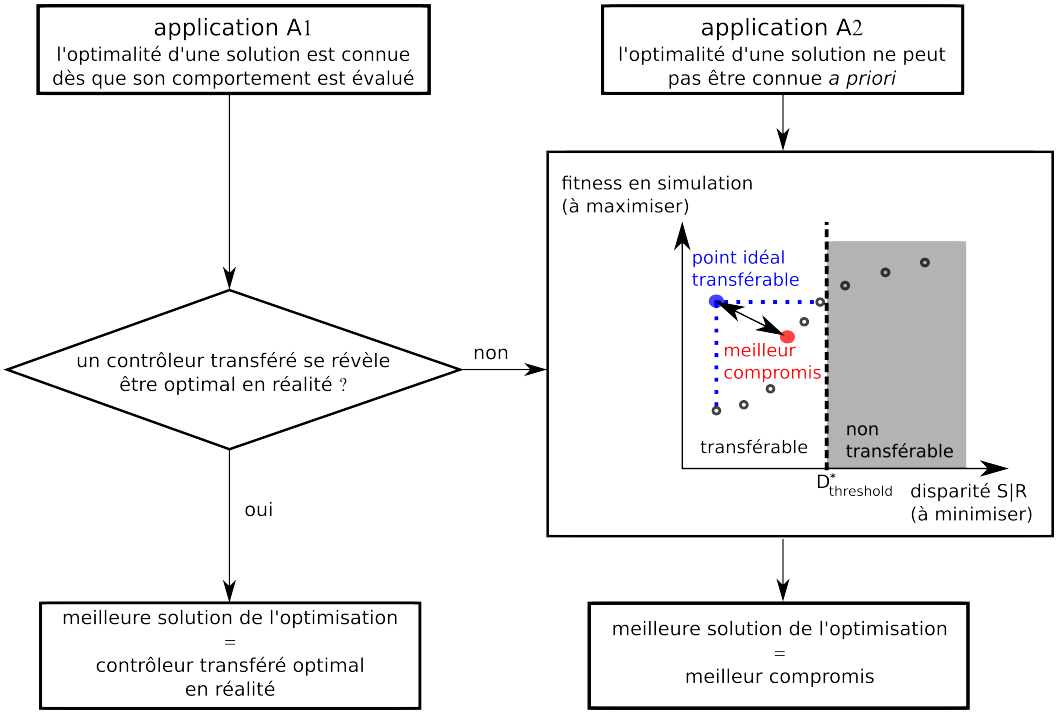

FIGURE 4.6 – Sélection de la meilleure solution : deux classes d'applications.

4.2.6 Validation sur deux applications robotiques

L'approche par transférabilité a tout d'abord été validée sur un des premiers dispositifs expérimentaux de Jakobi étudiant le problème de transfert de la simulation à la réalité (classe d'applications A1, [Jakobi 1997]). Cette application nous permet notamment de comparer notre approche à la méthodologie de Jakobi qui occupe une place importante dans la littérature. Une seconde expérience est ensuite menée sur un robot quadrupède à locomotion hybride roues-pattes doté de 8 degrés de liberté (classe d'applications A2, Fig. 4.13). Un tel robot peut implémenter divers types de comportements plus ou moins dynamiques et difficiles à simuler, donc plus ou moins transférables. Sur ces deux applications, l'approche par transférabilité est aussi comparée à des méthodes d'optimisation plus classiques : optimisation directe sur le robot, optimisation en simulation, optimisation en simulation suivie d'une recherche locale ou encore utilisation d'un modèle de substitution de la fitness construit sur l'espace des génotypes.

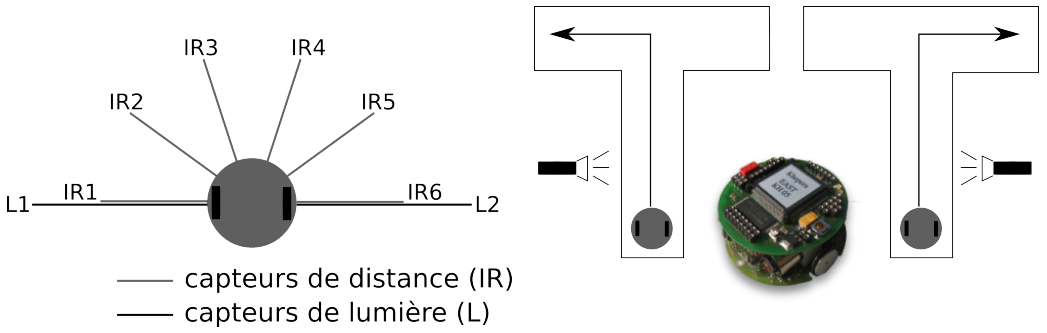

FIGURE 4.7 – A gauche, schéma des capteurs utilisés avec le robot mobile Khepera dans l'expérience originale. A droite, les deux cas de tests de l'expérience reproduite, ainsi que le robot Khepera.

4.3 Application 1 : navigation d'un robot e-puck

4.3.1 Dispositif expérimental

La première application envisagée vise à reproduire une des expériences de Jakobi sur le problème du passage de la simulation à la réalité [Jakobi 1997, Jakobi 1998a]. Nous nous intéresserons notamment à comparer l'approche par transférabilité avec l'approche des simulations minimales de Jakobi qui reste la méthodologie la mieux formalisée à l'heure actuelle traitant le problème de transfert à la réalité. Dans l'expérience originale, un robot disposant de deux roues navigue dans un labyrinthe en T et doit tourner à droite ou à gauche à la jonction suivant la position d'un stimulus lumineux reçu précédemment au milieu du premier corridor. Le robot dispose de 6 capteurs infrarouge de distance à l'avant et d'un capteur de lumière sur chaque côté. L'expérience comporte deux cas de tests : quand la lumière est sur la droite, le robot doit tourner à droite à la jonction et quand la lumière est à gauche, il doit tourner à gauche. Cette expérience de navigation simple est illustré sur la figure 4.7.

Au lieu du robot Khepera utilisé dans l'expérience de Jakobi, nous utilisons un robot mobile e-puck très similaire [Mondada *et al.* 2009]. Néanmoins, comme les capteurs de lumière du robot e-puck sont peu fiables, le dispositif expérimental implémenté diffère légèrement de la version originale. La détection de lumière est remplacée par la détection d'une zone de couleur en utilisant la caméra de l'e-puck [1]. Pendant le cas de test de gauche, le capteur visuel de gauche $V1$ est à 1 si le pixel en bas à gauche de la caméra est noir et le capteur visuel de droite $V2$ est toujours à 0. Pendant le cas de test de droite, le capteur visuel de droite $V2$ est à 1 si le pixel en bas à droite de la caméra est noir et le capteur de gauche est toujours à 0. Ce dispositif est illustré sur la figure 4.8.

D'après la méthodologie de Jakobi [Jakobi 1998a], une simulation minimale de ce dispositif expérimental doit être construite en découpant la tâche en deux parties : tout d'abord, le robot se déplace dans le premier corridor où il détecte la zone de

1. La résolution de la caméra est fixée à 40×40

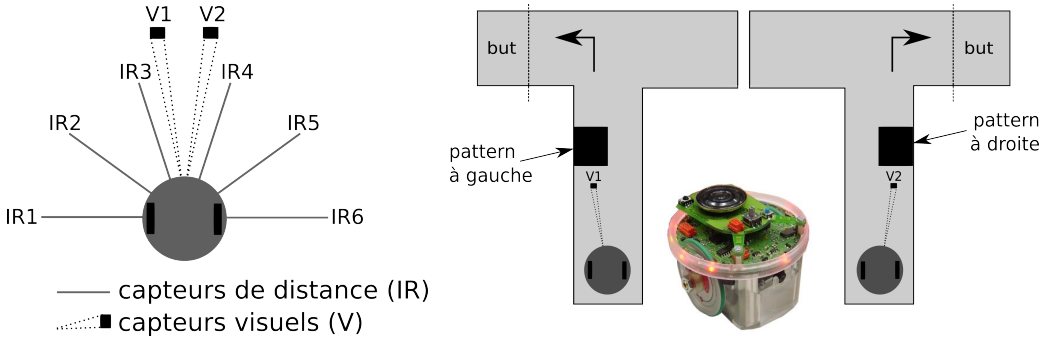

FIGURE 4.8 – A gauche, schéma des capteurs utilisés avec le robot mobile e-puck. A droite, les deux cas de tests de notre dispositif expérimental, ainsi que le robot e-puck.

couleur ; ensuite, quand le robot a parcouru une distance suffisante, il est téléporté dans le second corridor au niveau de la jonction, où il doit tourner dans la bonne direction. Etant données les capacités des capteurs du robot e-puck, cette simulation minimale diffère du labyrinthe en T réel, car il ne modélise pas exactement la jonction. Quand le robot est téléporté dans le second corridor, il ne devrait pas y avoir de mur derrière lui. Jakobi propose que si le robot détecte ce mur en simulation, les capteurs de distance doivent être suffisamment bruités pour qu'il ne puisse pas exploiter cette spécificité (voir figure 4.9, à gauche). L'implémentation de cette zone de bruit sera détaillée dans la prochaine section.

Le simulateur ne modélise ni friction ni glissement. La variation d'orientation du robot pendant un pas de temps est égale à la différence entre la distance parcourue par chacune des deux roues, divisée par le diamètre du robot (environ 75 mm).

Les capteurs de distance du robot sont modélisés comme suit : en fonction de la position du robot dans le labyrinthe simulé, la distance aux murs est calculée dans chacune des directions des 6 capteurs. Un modèle de la relation entre la distance aux murs d en millimètres et la valeur du capteur s [2] a été construit sur la base d'un échantillonnage des données des capteurs de l'e-puck utilisé pendant les expériences. Le modèle complet des capteurs est défini de la manière suivante [3] :

$$s = \begin{cases} 3300, & \text{si } d < 6mm \\ 3500 * F(d), & \text{si } 6mm \leq d \leq 60mm \\ 20, & \text{si } d > 60mm \end{cases}$$

La fonction F a été obtenue par régression polynomiale entre le logarithme des valeurs de capteurs et la distance aux murs. Un polynôme de degré 2 a été utilisé et la fonction F complète est définie par :

2. Les capteurs infrarouge de distance d'un robot e-puck prend des valeurs entre 0 et 3500.

3. Comme nous n'utilisons pas le même type de robot que dans [Jakobi 1997], le modèle de capteur est différent. De plus, la manière de construire le modèle utilisé dans le travail original n'est pas détaillée.

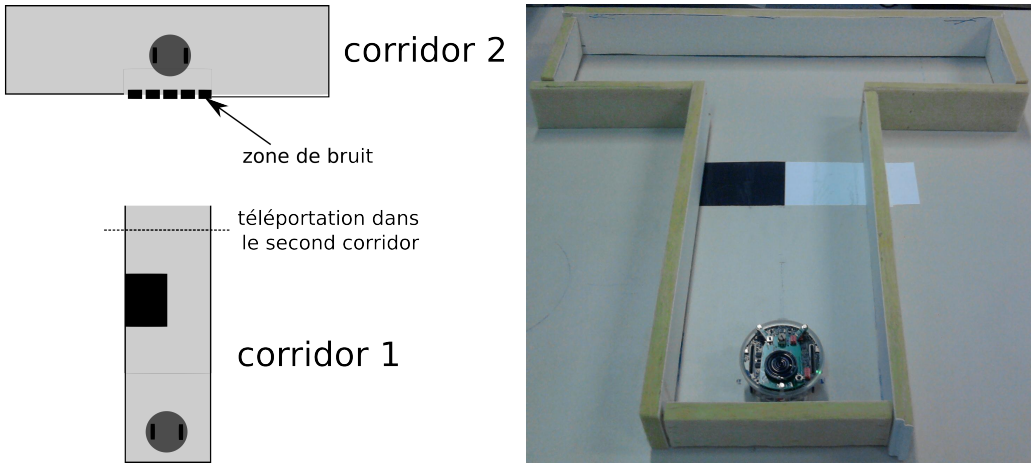

FIGURE 4.9 – A gauche, représentation de la simulation minimale du labyrinthe en T avec le pattern de couleur situé à gauche. A droite, photographie du labyrinthe réel dans le même cas de figure (la zone blanche indique la position du pattern dans l'autre cas de test quand il est situé à droite).

$$F(d) = \exp(0.6006882 - 0.1180289 * d + 0.0007215 * d^2)$$

Les valeurs de capteurs sont également bruitées par des déviations aléatoires uniformément distribuées dans l'intervalle $[-15,\ 15]$, puis divisées par 3500 avant d'être envoyées aux entrées du contrôleur.

Concernant le contrôle du robot, nous utilisons exactement la même structure de réseaux de neurones que dans l'expérience originale [Jakobi 1997]. Chaque réseau contient 10 neurones : 8 neurones sensoriels, chacun recevant de l'information d'un des capteurs (visuel ou de distance) et 2 neurones moteurs calculant la vitesse à appliquer aux roues. Chaque neurone a au plus 3 connexions sortantes. Afin d'imposer une symétrie bilatérale, le génotype n'encode que l'information codant pour la moitié des neurones, les 5 neurones correspondant aux capteurs et à la roue de gauche. Afin de développer le réseau complet, le demi-réseau est dupliqué suivant une symétrie axiale comme illustré sur la figure 4.10.

Les fonctions d'activation utilisées sont également les mêmes que dans l'expérience originale. Soit T_j le seuil du $j^{ème}$ neurone et soit w_{ij} le poids de la connexion partant du $i^{ème}$ neurone vers le $j^{ème}$ neurone, l'activation A_j du $j^{ème}$ neurone sensoriel est calculé par :

$$A_j = \begin{cases} 0, \text{ si } \sum A_i w_{ij} < T_j \\ 1, \text{ si } \sum A_i w_{ij} \geq T_j \end{cases}$$

Les neurones moteurs ont une fonction d'activation différente :

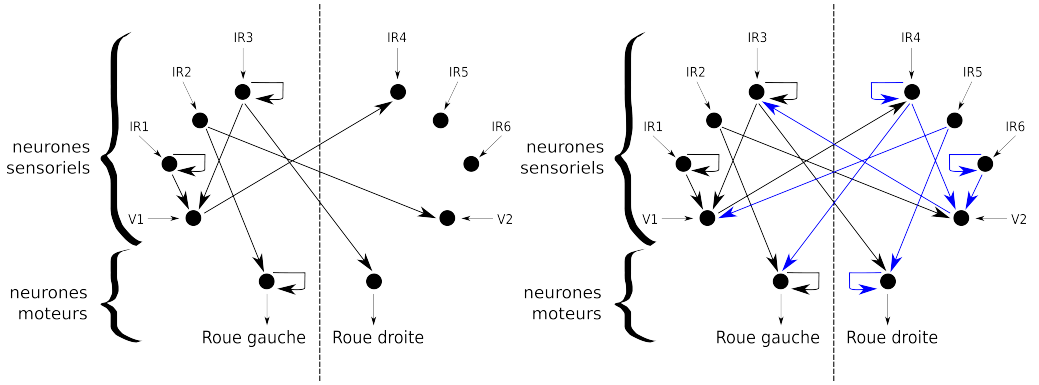

FIGURE 4.10 – A gauche, les connexions valides encodées dans le génotype. A droite, le réseau complet après développement.

$$A_j = \begin{cases} 0, & \text{si } \sum A_i w_{ij} - T_j < -1 \\ \sum A_i w_{ij} - T_j, & \text{si } -1 < \sum A_i w_{ij} - T_j \leq 1 \\ 1, & \text{si } \sum A_i w_{ij} - T_j > 1 \end{cases}$$

Le génotype encode 7 paramètres pour chacun des 5 neurones "de gauche" : (1) la valeur de seuil (16 valeurs discrètes distribuées uniformément entre -1 et 1) ; (2) le neurone de destination pour chacune des trois connexions sortantes possibles (valeurs entières entre 1 et 16) [4] ; (3) les poids correspondant à chacune de ces 3 connexions sortantes (16 valeurs discrètes régulièrement distribuées entre -2 et 2). Chaque neurone dépend donc de ces 7 paramètres et le génotype complet compte 35 valeurs de paramètres encodés sous la forme d'une chaîne de 140 bits (4 bits par paramètre).

Pendant l'étape d'évaluation, le comportement d'un individu est simulé pendant 150 pas de temps d'une position initiale fixe dans chacun des deux cas de test, lumière à gauche et lumière à droite. A chaque pas de temps, le réseau de neurones est mis à jour avec les valeurs des capteurs et les sorties des neurones moteurs sont multipliées par la vitesse maximale avant d'être appliquées aux deux roues. Pour chaque cas de test, la valeur de fitness est la distance de Manhattan (en millimètres) entre la position finale du robot et sa position initiale, plus un bonus de 1000 si le robot tourne dans la bonne direction à la jonction. Si le robot a bougé de d_x mm selon l'axe x et d_y mm selon l'axe y, la valeur de fitness correspondant au contrôleur c testé est donc :

$$fitness(c) = d_x + d_y + \begin{cases} 1000, & \text{si bon corridor à la jonction} \\ 0, & \text{sinon} \end{cases}$$

Les valeurs de fitness sont moyennées sur les deux cas de test afin de calculer la fitness globale d'un individu. La valeur maximale de fitness est environ 1700

4. Comme il n'y a que 10 neurones dans le réseau complet, les connexions dont la destination est entre 11 et 16 ne sont pas considérées lors du développement du génotype. Ceci autorise les neurones à posséder moins de 3 connexions sortantes.

mm. Dans le dispositif expérimental en réalité, le robot e-puck est contrôlé par un portable via une connexion Bluetooth et reçoit de nouvelles valeurs de vitesse toutes les 0.4 secondes pendant 150 pas de temps. Un cas de test dure donc 1 minute et l'expérience complète, 2 minutes. La vitesse maximale applicable aux roues est fixée à 2 centimètres par seconde. La vitesse maximale dans l'expérience originale était de 8 centimètres par seconde et les vitesses des roues étaient mises à jour 10 fois par seconde. Comme la connexion Bluetooth ne permet que 2.5 mises à jour par seconde, la vitesse maximale a été divisée par 4.

Pour les expériences dans le labyrinthe en T réel, la trajectoire 2D du robot est enregistrée avec deux scanners CODA cx1 (Charnwood Dynamics Ltd, UK) qui suivent le mouvement d'un marqueur posé sur le haut du robot.

4.3.2 Problèmes rencontrés lors de l'implémentation de l'approche de Jakobi

Afin d'obtenir des contrôleurs qui se transfèrent bien de la simulation à la réalité, la méthode originale de Jakobi consiste à chercher des individus suffisamment robustes en simulation. Ceux-ci devraient à la fois bien se transférer en réalité et aussi être robustes dans l'environnement réel. Dans cette partie, nous ne considérerons ici que le problème du transfert à la réalité. Les questions de robustesse seront abordées plus en détail dans le chapitre suivant.

Dans le problème du labyrinthe en T, les divergences entre la simulation et le dispositif réel sont liées aux capacités sensorielles du robot mobile e-puck. Les valeurs indiquées par les capteurs infrarouge de distance peuvent en effet beaucoup varier d'une expérience à l'autre et la durée pendant laquelle la zone de couleur est détectée par la caméra n'est pas non plus constante. Afin de faire face à ces divergences potentielles, des paramètres choisis de la simulation sont variés pendant l'optimisation d'une évaluation à l'autre afin d'éviter que les individus sur-apprennent un environnement particulier. Dans l'expérience originale de Jakobi, chaque individu est alors évalué dans 10 simulations dont les paramètres varient d'une génération à l'autre. Bien que ce mécanisme soit essentiel dans la méthodologie de la simulation minimale, trois points importants de l'approche ne sont pas détaillés dans les différents articles [Jakobi 1997, Jakobi 1998a].

- La quantité dont chaque paramètre doit varier d'une évaluation à l'autre n'est pas indiquée.
- L'implémentation de la zone de bruit utilisée dans la simulation minimale du second corridor est très rapidement décrite : dans cette zone, les capteurs de distance doivent renvoyer des valeurs parfois maximales, parfois faible, parfois aléatoires [Jakobi 1998a] [5].
- Enfin, il n'y a pas de critère pour déterminer quel est le meilleur individu de l'optimisation.

5. La phrase originale en anglais est : "sometimes [the infrared sensors] returned maximum values, sometimes low values, sometimes totally random values" [Jakobi 1998a].

Ce dernier point est important, car, chaque individu étant évalué dans des simulations potentiellement différentes, le meilleur individu à la génération n d'après la fitness n'est pas forcément le plus robuste : la fonction de fitness ne permet pas directement de déterminer si un individu est robuste ou non.

D'autre part, tous les paramètres semblent être variés de manière indépendante d'un individu à l'autre et d'une génération à l'autre dans l'approche originale, ce qui peut avoir des effets importants sur la fitness : par exemple, la taille du labyrinthe pouvant varier, des valeur de fitness obtenues dans des labyrinthes de grande taille sont potentiellement plus élevées que celles obtenues dans des petits labyrinthes. En pratique, les expériences préliminaires que nous avons effectuées avec de tels schémas d'optimisation n'ont pas permis de trouver des individus avec de hautes valeurs de fitness ou de haut niveau de robustesse en simulation avec le budget d'évaluations imposé.

4.3.3 Approches implémentées

4.3.3.1 Approche basée bruit, inspirée de celle de Jakobi

Afin d'obtenir des individus avec une bonne robustesse en simulation avec le budget d'évaluations autorisées (environ $2 \cdot 10^5$ évaluations), nous avons fait les choix suivants en réponse aux trois problèmes listés dans la partie précédente.

- Nous avons défini pour chaque paramètre à varier un ensemble de valeurs discrètes il pourra prendre et tous les individus sont évalués sur les dix même simulations.
- Les capteurs infrarouge de distance qui détectent la zone de bruit renvoient des valeurs aléatoires dans l'intervalle $[0,\ 3500]$.
- A la dernière génération, la meilleure solution est celle dont la valeur de fitness en simulation est maximale sur les dix simulations d'évaluation, bien que cette solution ne soit pas toujours celle de robustesse maximale.

Dans notre dispositif expérimental, les paramètres à varier sont : 1) la taille du labyrinthe (4 valeurs régulièrement distribuées entre 500 mm et 650 mm) ; 2) l'orientation initiale du robot (9 valeurs régulièrement distribuées entre -20° et 20°) ; 3) la longueur de la zone de couleur (5 valeurs régulièrement distribuées entre 20 mm et 120 mm). A chaque génération, les individus sont évalués dans 10 simulations choisies parmi les 180 possibles. Une des simulations correspond approximativement aux valeurs réelles des paramètres : 500 mm pour la taille du labyrinthe, 0° pour l'orientation initiale et 70 mm pour la longueur de la zone de couleur. Au début de chaque génération, une valeur est aléatoirement choisie pour chaque paramètre et pour chacune des 10 simulations dans l'ensemble de valeurs discrètes correspondant.

4.3.3.2 Approche par transférabilité

L'approche par transférabilité repose sur un processus d'optimisation multi-objectif basée sur la relation de dominance comme présentée dans la partie 2.2.

Les individus sont optimisés dans la simulation utilisant les valeurs réelles des paramètres : 500 mm pour la taille du labyrinthe, 0° pour l'orientation initiale et 70 mm pour la longueur de la zone de couleur. Avant de pouvoir implémenter l'approche, les deux fonctions suivantes doivent être définies : la mesure de disparité $S|R$ exacte et les caractéristiques comportementales utilisées pour la métrique *in silico* b_{dist}.

Soit $S_t = \{x_S^t,\ y_S^t\}$ et $R_t = \{x_R^t,\ y_R^t\}$, les positions horizontales du robot enregistrées respectivement en simulation et en réalité, soit $\overline{x_S}$ (resp. $\overline{y_S}$, $\overline{x_R}$, $\overline{y_R}$) la moyenne de x_S^t (resp. y_S^t, x_R^t, y_R^t), on note ε_x^c et ε_y^c les erreurs quadratiques moyennes normalisées suivantes (nMSE) :

$$\begin{cases} \varepsilon_x^c = \sum_{i=1}^{n} \frac{(x_S^i - x_R^i)^2}{\overline{x_S}\ \overline{x_R}} \\ \varepsilon_y^c = \sum_{i=1}^{n} \frac{(y_S^i - y_R^i)^2}{\overline{y_S}\ \overline{y_R}} \end{cases}$$

Nous définissons alors la valeur exacte $D^*(c)$ de disparité $S|R$ pour un contrôleur c donné comme suite :

$$D^*(c) = \varepsilon_x^c + \varepsilon_y^c$$

Comme indiqué précédemment, le nombre de pas de temps n est de 150. Nous choisissons empiriquement un seuil sur la valeur de disparité $S|R$ égal à $D^*_{threshold} = 0.1$: seuls les individus ayant une disparité $S|R$ inférieure à ce seuil seront considérés comme transférables (voir partie 4.2.5).

Pour construire la métrique *in silico*, 6 caractéristiques comportementales sont définies (3 pour le cas de test où la zone de couleur est à gauche et 3 pour l'autre cas de test) : 1) la distance de Manhattan parcourue par le robot pendant le cas de test correspondant ; 2) la distance minimale au mur gauche dans le premier corridor ; 3) la distance minimale au mur droit dans le premier corridor. Afin de calculer la métrique *in silico* b_{dist} entre deux individus en simulation, ces caractéristiques comportementales sont normalisées par leur valeur maximale respective en simulation : {800, 100, 100, 800, 100, 100} (en millimètres).

Le seuil de diversité est empiriquement fixé à $\tau_{div} = 0.25$ afin de mener environ 15 expériences de transfert en réalité pendant l'optimisation. Néanmoins, nous pouvons, dans cette expérience, déterminer l'optimalité d'un individu dès qu'il a été évalué, que ce soit en simulation ou en réalité (tâche de la classe A1 sur la figure 4.6). En effet, le but ici n'est pas de maximiser la fitness, mais de trouver un individu qui tourne dans la bonne branche du labyrinthe à la jonction. Ainsi, dès qu'un individu optimal en simulation se révèle également optimal en réalité lors d'une des expériences faites pendant l'optimisation avec une valeur de disparité $S|R$ suffisamment basse, on peut stopper le processus d'optimisation. Nous verrons que cette caractéristique de la tâche permet d'arrêter le processus d'optimisation relativement tôt, en moyenne après 6 expériences effectuées en réalité. D'autre part, afin de limiter le nombre d'individus transférés et de guider l'approximation de la disparité $S|R$ vers les zones potentiellement intéressantes de l'espace de recherche, ne sont transférés

en réalité que des individus optimaux en simulation [6].

4.3.3.3 Evolution directe sur le robot

A titre de comparaison, nous effectuons également une optimisation directement sur le robot physique. Chaque contrôleur est noté par la fitness obtenue quand il est transféré sur le robot physique, ainsi que par un objectif de diversité génotypique. La valeur de diversité est calculée comme la distance de Hamming moyenne entre le génotype de l'individu évalué (chaîne de 140 bits) et les génotypes du reste de la population. La taille de la population est de 4 et le nombre de générations est de 5, ce qui fait 20 expériences sur le robot par optimisation. Etant donné que cette approche nécessite plus d'expériences en réalité que les autres approches, elle n'a été répétée que 3 fois pour obtenir le même nombre d'expériences en réalité au total (environ 60 expériences par approche).

4.3.3.4 Modèle de substitution de la fitness

Un moyen classique d'optimiser des contrôleurs quand l'étape d'évaluation est trop longue ou coûteuse revient à construire un modèle de substitution de la fonction de fitness réelle [Jones 2001, Jin 2005]. Cette méthode ne s'appuie pas sur une simulation mais uniquement sur les informations générées en réalité par quelques expériences de transfert. Le modèle de substitution essaie d'interpoler la relation entre les paramètres de contrôle (souvent le génotype) et la fonction de fitness en réalité. Quand un modèle suffisamment précis est disponible, il suffit de maximiser la valeur de fitness prédite pour trouver la meilleure solution *a priori*.

L'utilisation d'un modèle de substitution requiert de définir la structure du modèle et une stratégie de mise à jour permettant de sélectionner les contrôleurs intéressants à transférer. Pour le problème du labyrinthe en T, on peut difficilement utiliser une interpolation par Kriging avec le budget d'évaluations proposé. En effet, un contrôleur dépend de 35 paramètres réels, ce qui nécessiterait donc au moins 71 expériences de transfert pour initialiser le modèle de covariance de Kriging. Pour cette application, nous considèrerons donc uniquement le cas où le modèle est interpolé par Inverse Distance Weighting (IDW).

Notre interpolation repose sur le même type de stratégie de mise à jour que pour l'approche par transférabilité. La sélection du prochain contrôleur à transférer est partie intégrante d'un processus d'optimisation multi-objectif, où une population de 200 individus est optimisée pendant 1000 générations suivant deux objectifs : la fitness prédite par le modèle de substitution et un objectif de diversité. Ceci permet de guider la population vers des individus à la fois potentiellement efficaces d'après le modèle et/ou nouveaux par rapport aux individus déjà transférés. A chaque génération, un contrôleur est sélectionné parmi les individus de la population dont la valeur de diversité est supérieure à un seuil pré-défini pour être ensuite

6. Exception faite de la première expérience de transfert, étant donné que l'individu correspondant est généré aléatoirement et donc probablement non optimal.

transféré sur le robot. Si aucun contrôleur n'a de diversité assez élevée, il n'y a pas de transfert pour cette génération.

La valeur de diversité est la distance de Hamming minimale entre le génotype de l'individu évalué et les génotypes des individus transférés précédemment au cours de l'optimisation. Soit $\mathcal{C}_T$ l'ensemble des contrôleurs déjà transférés, la valeur de diversité $diversity(c)$ correspondant au contrôleur c est donc :

$$diversity(c) = \min_{c_i \in \mathcal{C}_T} Hamming(c,\ c_i)$$

Le seuil de diversité est fixé à la main à $\tau_{div} = 0.55$, ce qui autorise environ 6 expériences de transfert sur le robot par optimisation. La meilleure solution trouvée est le contrôleur transféré dont la fitness est maximale.

4.3.3.5 Approche de référence

Pour l'approche de référence, chaque contrôleur est évalué par la fitness obtenue dans la simulation avec les paramètres réalistes. Il n'y a pas d'expérience de transfert durant l'optimisation et la meilleure solution trouvée est le contrôleur qui maximise la fitness en simulation.

4.3.3.6 Approche de référence suivie d'une recherche locale

Cette dernière approche commence par le même processus d'optimisation que l'approche de référence. Une fois que l'optimisation en simulation est terminée, la meilleure solution trouvée est utilisée comme point de départ d'une recherche locale sur le robot physique. La recherche locale itère le processus suivant 5 fois : la meilleure solution est mutée et si la version mutée obtient une meilleure fitness sur le robot, elle devient la nouvelle meilleure solution. Dans le cas où la meilleure solution trouvée en simulation est déjà optimale en réalité, il n'y a pas d'étape de recherche locale.

4.3.4 Résultats

La table 4.1 résume pour chaque approche où est menée l'étape d'évaluation (en simulation, en réalité, ou se servant d'informations issues des deux), ainsi que le nombre d'évaluations effectuées en moyenne sur le robot physique par optimisation.

Toutes ces approches utilisent l'algorithme évolutionniste multi-objectif NSGA-II [Deb *et al.* 2002b], qui repose sur une sélection élitiste par tournoi et une méthode de classement des individus basée sur la relation de dominance (voire partie 2.2.6.2 pour plus de détails). Pour les approches de référence ne comportant qu'un seul objectif, NSGA-II est équivalent à un algorithme évolutionniste élitiste par tournoi. Deux opérateurs sont définis sur le génotype : un opérateur de croisement et une mutation bit par bit de probabilité 0.7 (un individu muté subit en moyenne 2 inversions de bits). A l'exception de l'approche où l'évolution a directement lieu sur le

robot (4 individus, 5 générations, 3 répétitions), les autres approches sont répétées 10 fois avec une population de 200 individus optimisés pendant 1000 générations.

Ce travail a été implémenté dans le framework Sferes$_{v2}$ [Mouret & Doncieux 2010]. Le code source est disponible sur `http://www.isir.fr/evorob_db`. Des vidéos des comportements typiques obtenus avec l'approche de référence et l'approche par transférabilité sont disponibles sur `http://people.isir.upmc.fr/koos`.

Les résultats quantitatifs sont disponibles sur la figure 4.12 et dans la table 4.2. L'approche de référence trouve des individus à la fois optimaux en simulation et en réalité dans 3 cas sur 10. En moyenne, les meilleures solutions trouvées ont une performance de 1696 mm en simulation (σ = 3 mm) et de 832 mm en réalité (σ = 599 mm) avec une disparité moyenne de 0.25 (σ = 0.25). Ces résultats indiquent un problème de passage de la simulation à la réalité, car dans 7 cas sur 10, le meilleur individu trouvé est optimal en simulation, mais pas en réalité où il obtient une performance beaucoup plus faible. Néanmoins, la simulation minimaliste est suffisamment réaliste pour assurer *parfois* des transferts efficaces sans perte de performance. Un comportement typique obtenu avec l'approche de référence est représenté sur la figure 4.11 (deux colonnes de gauche).

Concernant l'approche de référence suivie d'une recherche locale, les meilleurs solutions trouvées après recherche locale ont des performances de 873 mm en moyenne en réalité (σ = 569 mm) avec une disparité moyenne de 0.26 (σ = 0.12). La recherche locale permet donc de trouver des individus plus performants en réalité avec une augmentation de fitness de 59 mm en moyenne (σ = 38 mm), valeur significativement plus grande que 0 d'après un test de Welch (p-value = $3 \cdot 10^{-3}$). Néanmoins, aucune de ces solutions n'est optimale sur le robot. Ainsi, une simple recherche locale ne permet pas de résoudre le problème du passage à la réalité dans ce cas.

Les deux approches où la phase d'évaluation se fait uniquement sur le robot physique, l'évolution directe sur le robot et le recours à un modèle de substitution de la fitness, obtiennent de mauvais résultats en réalité avec des performances moyennes respectives de 469 mm (σ = 45 mm) et 466 mm (σ = 77 mm). Les meilleures solutions pour ces deux approches présentent toujours le même comportement : le robot roule tout droit dans le premier corridor et reste bloqué contre le mur du fond dans le second corridor sans prendre en compte les zones de couleurs. Ceci montre à quel point le fait de limiter le nombre d'évaluations peut influer sur la qualité des résultats d'un processus d'optimisation : quelques évaluations ne permettent ici pas de trouver des comportements efficaces, étant donné le grand espace de recherche ($1.4 \cdot 10^{42}$ individus possibles).

Passons maintenant à l'approche basée bruit inspirée de la méthodologie de Jakobi. Les résultats présentés dans l'article original [Jakobi 1997] ne sont pas du tout reproduits. L'approche permet, ici, de trouver des individus à la fois optimaux en simulation et en réalité dans seulement 3 cas sur 10, alors que la méthode fonctionnait toujours dans l'expérience originale. En moyenne, les solutions trouvées ont une performance de 1383 en simulation (σ = 507 mm) et 1054 mm sur le robot e-puck (σ = 520 mm) avec une valeur de disparité $S|R$ moyenne de 0.20 (σ = 0.22). Ce n'est pas

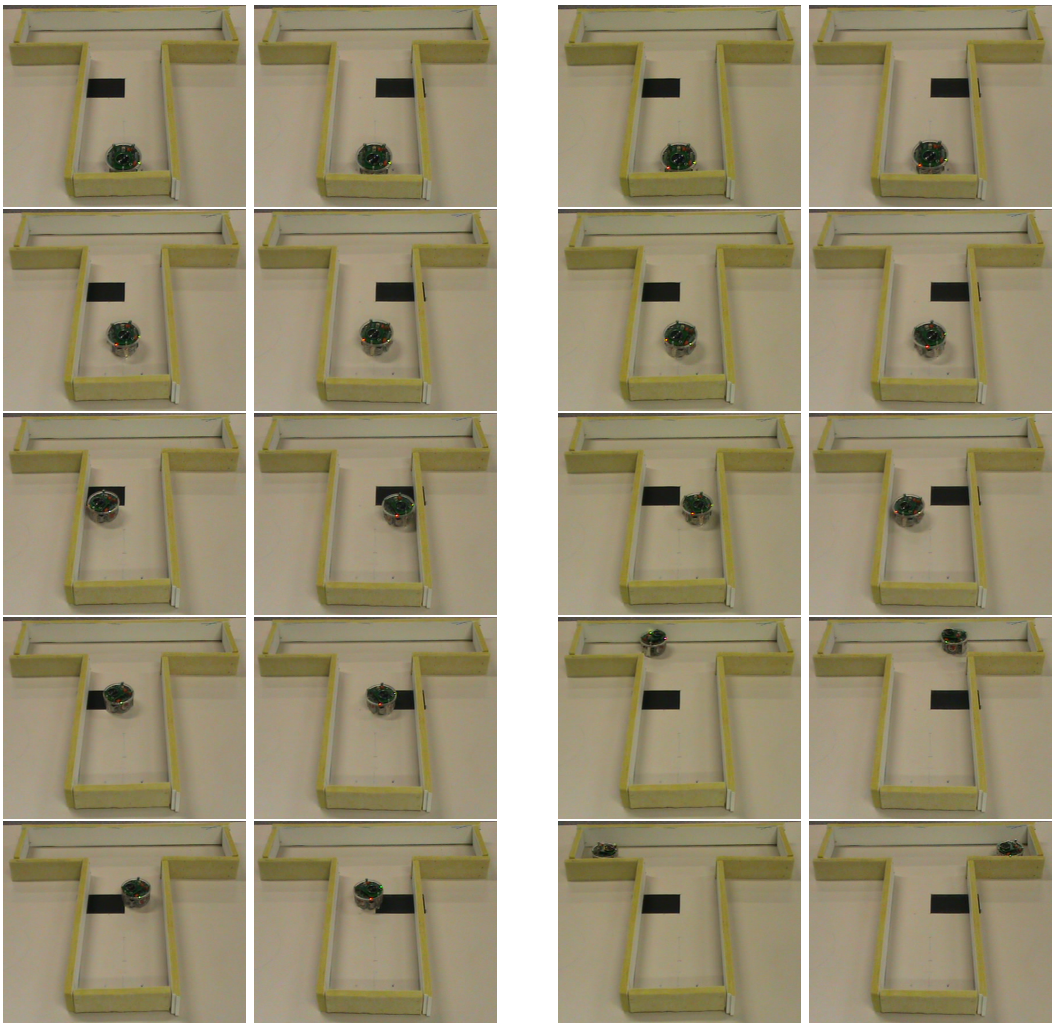

FIGURE 4.11 – A gauche, comportement typique évolué avec l'approche de référence : le comportement transféré est inefficace et le robot ne résout pas la tâche, car incapable d'éviter les murs. A droite, comportement typique obtenu avec l'approche par transférabilité. Le robot se comporte de manière optimale dans les deux cas de tests. Pour chaque couple de colonnes, la colonne de gauche (resp. de droite) correspondant au cas de test où la zone de couleur est à gauche (resp. à droite).

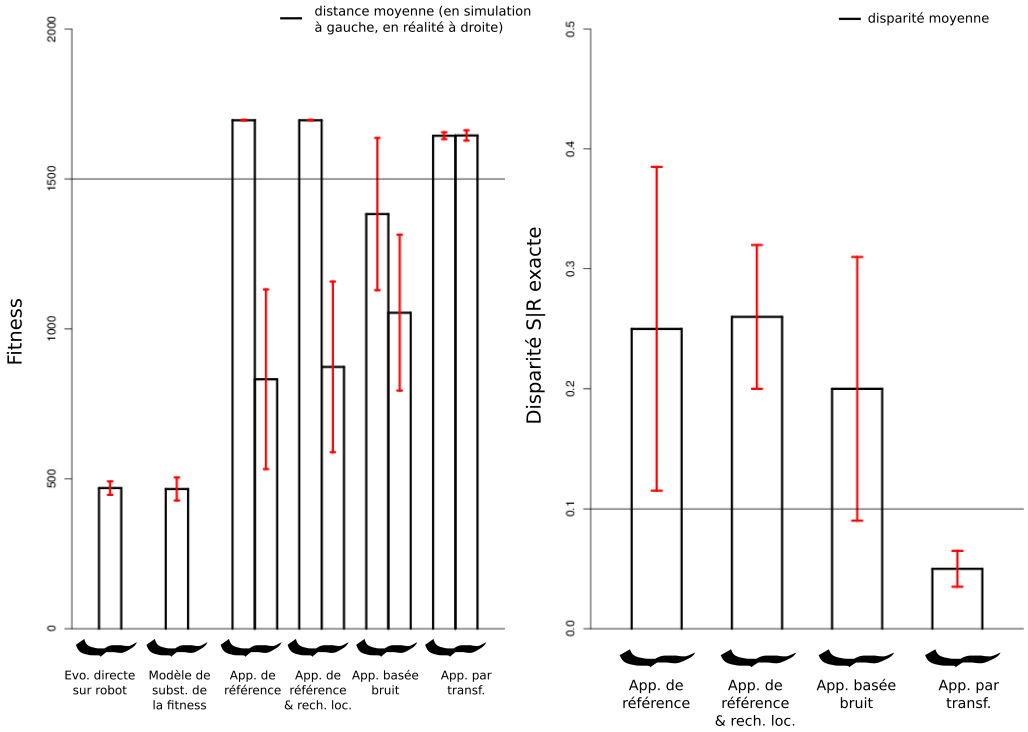

FIGURE 4.12 – A gauche, fitness moyenne des meilleures solutions trouvées avec toutes les approches. Chaque couple de barres indique la fitness en simulation à gauche et la fitness en réalité à droite, sauf pour les approches d'évolution directe sur le robot et du modèle de substitution de la fitness où seule la fitness en réalité est indiquée. A droite, les valeurs de disparité $S|R$ exactes correspondant à ces solutions. Moyennes et écarts types sont calculés sur 10 répétitions, sauf pour l'évolution directe sur le robot qui n'a été répétée que 3 fois. Les barres d'erreur indiquent une unité d'écart-type. L'approche par transférabilité obtient de meilleurs résultats en trouvant toujours des contrôleurs à la fois transférables (valeurs de disparité $S|R$ inférieures à $D^*_{threshold} = 0.1$) et optimaux en réalité sur le robot e-puck.

Approches	Evaluation		Nombre d'expériences sur le robot physique
	simulation	réalité	
Evo. directe sur robot		x	20
Modèle de subst. de la fitness IDW		x	6 (mean, $\sigma = 1$)
App. référence	x		0
App. référence & rech. loc.	x	x	6
App. basée bruit	x		0
App. par transf.	x	x	6 (mean, $\sigma = 2$)

TABLE 4.1 – Localisation de l'étape d'évaluation : totalement en simulation, totalement en réalité, ou dans les deux à la fois. Le nombre moyen d'expériences de transfert effectuées sur le robot pendant l'optimisation est indiqué pour chacune des approches.

totalement surprenant, étant donnés les nombreux détails manquants pour pouvoir ré-implémenter précisément l'approche : certains de nos choix ont vraisemblablement pu mener à ces résultats médiocres. Néanmoins, l'approche basée bruit trouve des solutions plus performantes en moyenne que les deux approches de référence. D'autre part, des résultats récents sur un dispositif expérimental proche suggèrent que ce genre d'approches nécessite un budget d'évaluations plus important en simulation pour converger vers des solutions effectivement robustes [Pinville *et al.* 2011]. Ces résultats concernant les capacités de généralisation des contrôleurs seront détaillés dans le chapitre 5.

L'approche par transférabilité fonctionne bien sur les 10 répétitions en obtenant des solutions dont la disparité moyenne est faible (0.05 en moyenne, $\sigma = 0.03$) et ce, sans perte de performance significative lors du transfert à la réalité. Les solutions trouvées ont des performances moyennes de 1644 mm ($\sigma = 23$ mm) en simulation et 1645 mm en réalité ($\sigma = 35$ mm). Un comportement typique obtenu avec cette approche est représenté sur la figure 4.11 (deux colonnes de droite). L'approche surpasse clairement celle recourant à un modèle de substitution de la fitness réelle. Ceci suggère qu'interpoler la fonction de fitness réelle de zéro nécessite un grand nombre d'expériences supplémentaires.

Un autre résultat intéressant (voir table 4.2) concerne le temps de calcul moyen : alors qu'effectuer des expériences sur le robot devrait ralentir le processus d'optimisation, il permet au contraire de résoudre la tâche en réalité en quelques générations (environ 52 générations, moins d'1 heure) avec seulement 6 expériences de transfert sur le robot e-puck en moyenne par optimisation (voir table 4.1). En effet, comme l'optimalité d'un individu est connu dès que celui-ci est évalué dans le labyrinthe en T, l'algorithme peut être stoppé dès qu'un individu transféré en réalité se révèle être optimal. Ce résultat n'est évidemment pas généralisable à des tâches plus réalistes où le comportement optimal n'est pas connu *a priori*.

Le pourcentage de simulations "réussies" indiqué dans la table 4.2 correspond

Approches	Temps de calcul	Taux de succès	Pourcentages de simulations "réussies"
App. référence	$\sim$ 4h	30%	27%
App. référence + rech. loc.	$\sim$ 4.25h	30%	27%
App. basée bruit	$\sim$ 12h	30%	83%
App. par transf.	$\sim$ 0.75h	100%	19%

TABLE 4.2 – Temps de calcul et taux de succès pour chacune des approches sur les 10 répétitions. Le pourcentage de simulations "réussies" correspond au pourcentage moyen de simulations parmi les 180 combinaisons possibles des valeurs de paramètres où les solutions trouvées par chaque approche se comportent de manière optimale.

à la proportion des 180 simulations possibles qui sont résolues en moyenne avec les meilleures solutions obtenues dans les différentes approches, c'est-à-dire où ces solutions se comportent de manière optimales et effectuent la tâche. Ce critère est intéressant, car on remarque notamment que l'approche basée bruit trouve des contrôleurs optimaux en moyenne dans 83% des 180 simulations possibles, ce qui prouve que le processus d'optimisation trouve bien des individus robustes, même si aucun n'est optimal dans toutes les simulations. Quoiqu'il en soit, il n'y a pas de lien clair entre cette valeur de robustesse et les problèmes de transfert à la réalité rencontrés.

Les valeurs obtenues avec les approches de référence et l'approche par transférabilité sont plus faibles, respectivement 27% et 19%. Ainsi, les meilleures solutions trouvées avec l'approche par transférabilité n'auraient pas été sélectionnées par l'approche basée bruit. En d'autres termes pour ce dispositif expérimental, des comportements "robustes" ne sont pas forcément transférables et des comportements transférables ne présentent pas systématiquement des caractéristiques particulières en terme de robustesse.

En conclusion pour cette application, l'approche par transférabilité permet de résoudre le problème du passage de la simulation à la réalité, en trouvant notamment de meilleures solutions que l'approche basée bruit inspirée de la méthodologie de Jakobi ou que d'autres approches plus classiques.

4.4 Application 2 : marche d'un robot quadrupède

La robotique évolutionniste a souvent été appliquée à des problèmes de locomotion. En particulier, la marche quadrupède présente l'intérêt de mettre en jeu plusieurs types de comportements : des marches statiques et faciles à modéliser, des comportements très dynamiques et plus complexes, ... Comme ces comportements de marche ne nécessitent pas le même degré de précision pour être modélisées correctement, on s'attend donc à observer différentes qualités de transfert sur le robot suivant le modèle de simulation utilisé. Afin d'exploiter une telle variété de comportements, notre seconde application consiste à trouver un comportement de marche aussi rapide que possible sur un robot quadrupède à 8 degrés de liberté présenté sur

FIGURE 4.13 – A gauche, le robot utilisé pour cette application, construit d'après le robot Hylos (photographie de droite).

la figure 4.13 (image de gauche). La fonction de fitness sera simplement la distance parcourue par le robot en un temps donné. Contrairement à l'application précédente, il est important de noter que nous ne connaissons pas l'optimum (application de classe A2, voir figure 4.6), c'est-à-dire la vitesse maximale atteignable par le robot avec le type de contrôleurs utilisé. La fitness sera donc maximisée sans *a priori*.

4.4.1 Robot et dispositif expérimental

Le robot physique a été construit à l'aide d'un Kit de Bioloid d'après le robot hybride roues-pattes Hylos [Grand *et al.* 2004a] (voir figure 4.13, image de droite), initialement conçu pour l'exploration autonome de terrains accidentés (volcanique par exemple). Notre robot en est un modèle réduit qui reprend essentiellement sa morphologie et ses degrés de liberté. Chacune des quatre pattes du robot est composée de 4 actuateurs Dynamixel AX-12+. Comme dans cette application, nous ne considérons que le problème de la marche quadrupède, seuls les 2 moteurs les plus hauts d'une patte sont contrôlés. Les autres (le troisième moteur et la roue terminale) restent fixes. Chaque patte contient donc un moteur supérieur et un moteur inférieur comme indiqué sur la figure 4.14. Le contrôle se fait en position et la vitesse des moteurs est automatiquement asservie par un micro-contrôleur intégré en fonction de la différence entre la position actuelle et la position désirée (pour plus de détails, se référer à la documentation des moteurs Dynamixels AX-12+). La vitesse maximale des moteurs est de 1.75 rad/s. Pendant les expériences, le robot est alimenté par un câble et contrôlé par l'intermédiaire d'un module USB2Dynamixel connecté à un portable via un câble USB.

Pour construire le modèle de simulation, nous utilisons le moteur open-source Bullet Physics Library, qui s'avère être à la fois rapide et relativement précis comparé à d'autres moteurs physiques habituellement utilisés [Boeing & Bräunl 2007]. Tous les paramètres de la simulation sont laissés aux valeurs par défaut proposées par la bibliothèque. Le sol est simulé avec une friction non nulle. Le moteur physique utilise un solveur de contraintes basé sur la méthode de Gauss-Seidel afin de gérer les collisions et les contraintes sur les articulations (nombre maximum d'itérations

de 200, plus de détails sont disponibles dans la documentation de Bullet : http://bulletphysics.org).

Deux phénomènes principaux diffèrent entre le modèle de simulation et l'environnement du robot physique, ce qui peut entraîner des problèmes de passage de la simulation à la réalité.

- Le glissement observé sur le sol physique n'est pas modélisé avec précision.
- Des comportements instables peuvent être notés avec des fitness élevées non réalistes en simulation, notamment à cause des contacts entre les différentes parties des pattes.

Pour cette application, les points suivants ont été modélisés avec attention : les dimensions du robot, les masses des différents éléments, l'asymétrie de masse du corps principal du robot, la zone de contact des roues et l'asservissement des vitesses des moteurs pour le micro-contrôleur intégré (en fonction des données issues de la documentation des moteurs Dynamixel AX-12+). Le robot simulé se compose de 14 corps rigides et de 8 contraintes d'articulations pour modéliser les moteurs. A noter que pour une telle application, la méthodologie de Jakobi peut difficilement être envisagée. Il est en effet difficile de définir un ensemble de paramètres pertinents à faire varier en simulation pour obtenir des comportements robustes. Nous n'avons donc pas appliqué la méthode de Jakobi à cette application-ci.

Les expériences effectuées en simulation et en réalité suivent le même déroulement. Au début de chaque expérience, toutes les positions angulaires des moteurs sont fixées à 0. Des nouvelles positions sont alors envoyées aux moteurs toutes les 0.1 secondes pendant 10 secondes et la trajectoire 2D du centre géométrique du robot dans le plan horizontal est enregistrée avec une fréquence de 20 Hz (200 points au total). Une fois que l'évaluation est terminée, la position initiale du robot est soustraite à tous les points afin de faire commencer la trajectoire du robot au point (0, 0) et de ne pas dépendre d'une position initiale variable lors des comparaisons de trajectoires.

Concernant les expériences en réalité, la position 2D du robot est enregistrée à l'aide de 3 scanners CODA cx1 (Charnwood Dynamics Ltd, UK) qui détectent la position absolue de marqueurs disposés sur le corps du robot dans un repère défini par l'utilisateur. Pour obtenir la position du centre géométrique, deux marqueurs sont utilisés, un à l'avant et un à l'arrière comme indiqué sur la figure 4.14. Leurs positions sont ensuite moyennées pour obtenir les coordonnées du centre.

Afin d'étudier le problème du passage de la simulation à la réalité, nous utilisons un contrôleur très simple : la position de chaque moteur est contrôlée par un signal sinusoïdal. Les 8 fonctions sinusoïdales contrôlant les différents moteurs ne dépendent que des 2 même paramètres réels p_1 et p_2, avec $(p_1,\ p_2) \in [0,1]^2$. La position angulaire désirée α^d du moteur i à l'instant t est calculé comme suit :

$$\alpha^d(i,t) = \frac{5\pi}{12} * dir(i) * p_1 - \frac{5\pi}{12} * p_2 * sin(2\pi t - \phi(i))$$ [7]

7. Les positions angulaires des actuateurs Dynamixel AX-12+ varient dans l'intervalle $[-\frac{5\pi}{12}, \frac{5\pi}{12}]$.

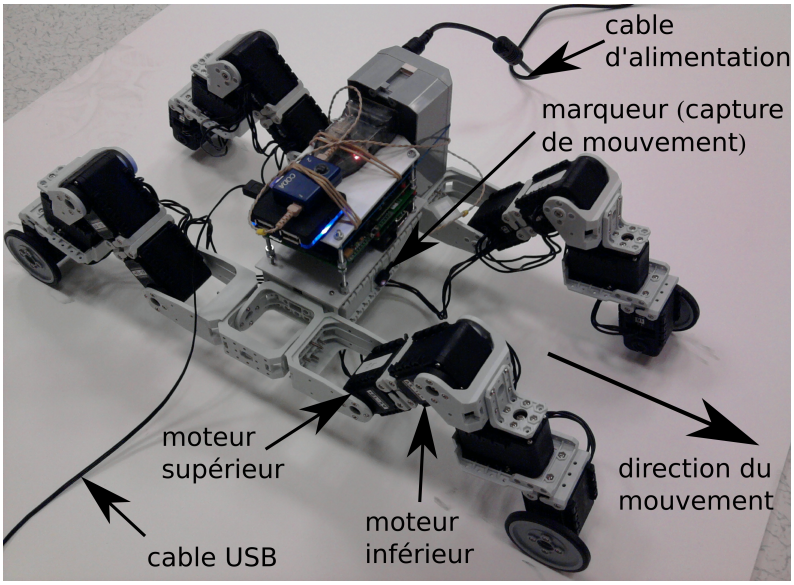

FIGURE 4.14 – Robot quadrupède utilisé dans notre dispositif expérimental.

$dir(i)$ vaut 1 pour les moteurs des pattes avant-droite et arrière-gauche, -1 sinon (voir figure 4.14 pour l'orientation). La phase angulaire $\phi(i)$ vaut 0 pour les moteurs supérieurs et $\pi/2$ pour les moteurs inférieurs. Les deux moteurs d'une même patte sont donc contrôlés avec la même fonction sinusoïdale avec un décalage de phase de $\pi/2$. Pour simplifier, p_1 fixe la pose initiale du robot et p_2 l'amplitude maximale des mouvements des pattes.

La fonction de fitness est la distance parcourue par le robot en 10 secondes. Bien qu'il n'y ait que deux paramètres à optimiser, le paysage de fitness en simulation est complexe comme indiqué sur la figure 4.15. De telles variations de fitness indiquent une relation fortement non-linéaire entre l'espace des génotypes et l'espace des performances pour cette application.

4.4.2 Approches implémentées

4.4.2.1 Approche par transférabilité

La mesure de disparité $S|R$ exacte est basée sur la trajectoire du robot dans le plan horizontal et plus exactement sur la distance parcourue $\sqrt{x_t^2 + y_t^2}$ au cours du temps à partir du point d'origine. Soit S_t et R_t, les distances parcourues respectivement en simulation et en réalité au pas de temps t ($t \in [1,\ 200]$), soit $\overline{S}$ (resp. $\overline{R}$) la moyenne des S_t (resp. des R_t), la valeur exacte $D^*(c)$ de disparité $S|R$ correspondant au contrôleur c évalué est l'erreur quadratique moyenne normalisée (nMSE) entre les suites des S_t et des R_t :

$$D^*(c) = \sum_{t=1}^{200} \frac{(S_t - R_t)^2}{\overline{S}\ \overline{R}}$$

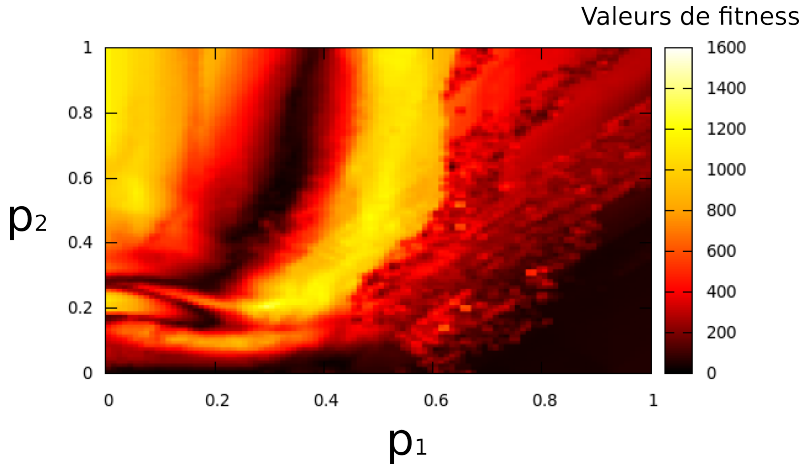

FIGURE 4.15 – Paysage exhaustif de fitness en simulation pour l'application de marche quadrupède : distance parcourue en simulation associée à chaque couple de paramètres (p_1, p_2) (environ $2.6 \cdot 10^5$ contrôleurs possibles). Les valeurs des paramètres de contrôle p_1 et p_2 sont respectivement indiquées suivant l'axe x et l'axe y. La couleur représente l'intensité de la valeur de fitness.

Cette mesure de disparité $S|R$ permet d'évaluer avec précision les variations entre la trajectoire réelle et la trajectoire simulée correspondant à un même contrôleur. Le contrôleur est d'autant plus transférable que ces trajectoires sont proches.

Il faut également définir les caractéristiques comportementales utilisées pour la métrique *in silico* b_{dist} qui intervient à la fois dans le modèle de substitution de la disparité $S|R$, et dans le calcul des valeurs de diversité. Nous définissons les 3 caractéristiques comportementales suivantes, évaluées uniquement en simulation pour chaque contrôleur :

- la distance parcourue pendant l'expérience (i.e. la fitness) ;
- la hauteur moyenne du centre géométrique du robot ;
- l'orientation angulaire finale du robot dans le plan horizontal.

Avant d'utiliser la métrique, les valeurs de ces caractéristiques comportementales sont normalisées par leurs valeurs maximales observées en simulation, respectivement $\{1.5, 0.2, 3.14\}$. Afin d'autoriser seulement 10 transferts en moyenne pendant l'optimisation, nous fixons de manière empirique la valeur du seuil de diversité à $\tau_{div} = 0.1$.

4.4.2.2 Evolution directe sur le robot

A titre de comparaison et comme dans l'application précédente, nous avons mené des optimisations directement sur le robot physique. Chaque contrôleur est donc évalué par la distance parcourue en réalité et un objectif de diversité génotypique. La valeur de diversité utilisée est la distance euclidienne moyenne entre le génotype de l'individu considéré (vecteur des 2 paramètres réels p_1 et p_2) et les génotypes du reste de la population. La taille de la population est de 4 et le nombre de générations

de 5, ce qui signifie 20 expériences sur le robot pendant l'optimisation. Etant donné que cette approche nécessite plus d'expériences que les autres, elle n'a été répétée que 5 fois pour avoir la même quantité globale d'expériences en réalité au total (environ 100 expériences par approche).

4.4.2.3 Modèle de substitution de la fitness

Au lieu de recourir à un simulateur physique, il est possible de construire directement un modèle de substitution de la fonction de fitness réelle [Jones 2001, Jin 2005]. Le modèle va chercher à interpoler la relation entre l'espace des génotypes et l'espace des performances en réalité. Dès qu'un tel modèle est disponible, la meilleure solution trouvée est celle qui maximise la prédiction de la fitness d'après le modèle.

Pour utiliser un modèle de substitution, il est nécessaire de définir à la fois la structure du modèle et la stratégie de mise à jour qui va servir à sélectionner les contrôleurs à transférer sur le robot afin d'améliorer la prédiction. Contrairement à l'application de navigation précédente où le génotype comportait un grand nombre de paramètres, seuls 2 paramètres sont optimisés. L'utilisation d'un modèle de Kriging est donc envisageable ici et nous avons implémenté deux approches suivant la structure du modèle d'interpolation utilisé, soit par IDW, soit par Kriging.

Modèle par Inverse Distance Weighting Pour cette variante, nous avons choisi une stratégie de mise à jour similaire à celle utilisée dans l'approche par transférabilité. La sélection du contrôleur à transférer est incluse dans un processus d'optimisation multi-objectif où 40 individus sont évolués pendant 100 générations suivant 2 objectifs : la fitness prédite par le modèle et un objectif de diversité génotypique. La population est donc guidée vers les zones de l'espace de recherche plutôt efficaces d'après le modèle et/ou plutôt nouvelles. A chaque génération, les contrôleurs dont la valeur de diversité est supérieure à un seuil τ_{div} sont considérés et l'un deux est aléatoirement sélectionné pour être transféré sur le robot. Le modèle est ensuite mis à jour avec les nouvelles données générées en réalité. Si aucun contrôleur de la population n'a une valeur de diversité suffisante, aucune expérience de transfert n'est effectuée pendant cette génération.

L'objectif de diversité est la distance euclidienne minimale entre le vecteur des 2 paramètres de contrôle de l'individu évalué et les vecteurs de paramètres correspondant aux contrôleurs ayant déjà été transférés sur le robot. Ainsi, soit $\mathcal{C}_T$ l'ensemble des contrôleurs déjà transférés en réalité, la valeur de diversité génotypique $diversity(c)$ correspondant au contrôleur c est calculée comme suit :

$$diversity(c) = \min_{c_i \in \mathcal{C}_T} \sqrt{(p_1^c - p_1^{c_i})^2 + (p_2^c - p_2^{c_i})^2}$$

Le seuil de diversité est fixé à $\tau_{div} = 0.075$ afin d'autoriser 10 expériences de transfert en moyenne pendant l'optimisation.

Modèle par Kriging Comme il n'y a que 2 paramètres de contrôle pour cette application, il est possible d'utiliser un modèle de Kriging sans avoir à effectuer de trop nombreuses expériences pour initialiser le modèle. Notre implémentation est essentiellement basée sur celle décrite dans l'article [Hemker *et al.* 2006] et l'optimisation n'est pas assurée par un processus évolutionniste. L'initialisation du modèle de Kriging nécessite ici 5 expériences préliminaires et les différentes répétitions effectuées avec cette structure de modèle démarrent toutes du même modèle initial. Les 5 expériences préliminaires effectuées (voir figure 4.17) correspondent aux vecteurs de paramètres de contrôle suivant : (0, 0), (1, 0), (0, 1), (1, 1), (0.5, 0.5).

La stratégie de mise à jour est également inspirée de celle présentée dans l'article [Hemker *et al.* 2006] et compte 3 étapes.

- 10^4 vecteurs de paramètres de contrôle sont générés aléatoirement afin de trouver, d'après le modèle substitution courant, le contrôleur c_{maxfit} qui maximise la prédiction de fitness et le contrôleur c_{maxerr} qui maximise l'erreur quadratique moyenne (MSE) attendue.
- Si c_{maxfit} n'est pas trop proche d'un contrôleur déjà transféré auparavant, il est transféré à son tour. Dans le cas contraire, c_{maxerr} est transféré à sa place.
- Le modèle de substitution est mis à jour avec la nouvelle valeur de fitness exacte obtenue.

Cette stratégie de mise à jour est itérée 10 fois pendant une optimisation, ce qui implique 10 expériences de transfert. Le contrôleur c_{maxfit} n'est alors transféré sur le robot que si sa distance euclidienne aux individus ayant déjà été transférés est toujours supérieure à une borne $\varepsilon = 0.025$. Dans le cas contraire, l'individu c_{maxerr} est transféré.

L'approche par Kriging est répétée 10 fois. La meilleure solution de l'optimisation est le contrôleur transféré qui a obtenu la meilleure fitness réelle. L'implémentation et l'utilisation du modèle de Kriging est basée sur la toolbox DACE de Matlab [Lophaven *et al.* 2002].

4.4.2.4 Approche de référence

L'approche de référence met en jeu un algorithme évolutionniste, où la performance des contrôleurs est évaluée uniquement en simulation. Chaque contrôleur est alors noté par la distance parcourue en simulation et aucune expérience de transfert n'a lieu pendant le processus. La meilleure solution est celle qui maximise la fitness en simulation.

4.4.2.5 Approche de référence & recherche locale

Comme pour l'application précédente, cette approche s'effectue en deux phases : les contrôleurs sont tout d'abord optimisés en simulation comme avec l'approche de référence simple ; ensuite, une recherche locale est effectuée sur le robot physique avec comme point de départ le meilleur individu trouvé précédemment en simulation. La recherche locale itère le processus suivant 10 fois : la meilleure solution est mutée et

si la version mutée obtient une meilleure fitness sur le robot, elle devient la nouvelle meilleure solution.

4.4.2.6 Approche de référence & diversité comportementale

La dernière approche implémentée pour cette application vise à optimiser 2 objectifs : la distance parcourue en simulation et un objectif de diversité comportementale. Cet objectif de diversité est calculé en comparant le comportement de l'individu considéré à une archive d'individus sélectionnés durant l'optimisation. La distance entre individus utilisée est la métrique *in silico* b_{dist} décrite pour l'approche par transférabilité. Soit C_s l'archive de contrôleurs, la valeur de diversité comportementale $diversity(c)$ pour un contrôleur c donné est :

$$diversity(c) = \min_{c_i \in \mathcal{C}_S} b_{dist}(c,\ c_i)$$

L'archive C_s est construite comme suit : à chaque génération du processus d'optimisation, si des contrôleurs ont une valeur de diversité supérieure à un seuil τ_{div}, l'un d'eux est ajouté aléatoirement à l'archive. La valeur de τ_{div} est fixée à 0.075 afin d'obtenir environ 10 individus dans l'archive à la fin de l'optimisation. Ce calcul de diversité est en fait identique à celui effectué avec l'approche de transférabilité, mais sans s'accompagner de la construction d'un modèle de substitution de la disparité $S|R$ et donc sans expérience de transfert sur le robot.

4.4.3 Résultats

La table 4.3 résume pour chaque approche où est menée l'étape d'évaluation (en simulation, en réalité, ou se servant d'informations issues des deux), ainsi que le nombre d'évaluations effectuées en moyenne sur le robot physique par optimisation.

Comme pour la précédente application, toutes les approches, exceptée celle utilisant un modèle de Kriging, utilisent l'algorithme évolutionniste multi-objectif NSGA-II [Deb *et al.* 2002b] (voir partie 2.2.6.2 pour plus de détails sur cet algorithme). Le génotype est le vecteur des 2 paramètres de contrôle $(p_1,\ p_2) \in [0,\ 1]^2$. Deux opérateurs sont définis sur ce génotype : une mutation ponctuelle gaussienne de moyenne nulle et d'écart-type 0.2 et un opérateur de croisement. La probabilité de mutation est de 0.5. Hormis pour l'évolution directe sur le robot physique (4 individus, 5 générations, répétée 5 fois) et l'approche par Kriging (pas de processus évolutionniste, répétée 10 fois), les différentes approches ont été répétées 10 fois avec une population de 40 individus pendant 100 générations. Les résultats obtenus avec les différentes approches sont indiqués sur la figure 4.18.

Ce travail a été implémenté dans le framework Sferes_{v2} [Mouret & Doncieux 2009a]. Le code source est disponible sur `http://www.isir.fr/evorob_db`. Des vidéos des comportements typiques obtenus avec l'approche de référence et l'approche par transférabilité sont disponibles sur `http://people.isir.upmc.fr/koos`.

4.4.3.1 Résultats obtenus avec les approches de référence

Les meilleures solutions obtenues avec l'approche de référence simple obtiennent des performances de 1294 mm en moyenne en simulation ($\sigma = 55$ mm) et 411 mm en moyenne sur le robot ($\sigma = 425$ mm). Les valeurs de disparité $S|R$ correspondantes sont de 4.90 en moyenne avec un écart-type de 8.78. Un comportement typiquement obtenu avec l'approche de référence est présentée sur la figure 4.20 (colonne de gauche). Globalement, 3 des meilleures solutions trouvées sur 10 se transfèrent bien sur le robot physique avec de hautes valeurs de performance, et 7 se transfèrent mal. On peut en tirer deux conclusions.

- Il y a un problème de passage de la simulation à la réalité en optimisant directement avec ce modèle de simulation.
- Néanmoins, le simulateur physique est suffisamment réaliste et permet *parfois* de trouver des individus performants qui se transfèrent bien.

L'approche de référence suivie d'une recherche locale permet de trouver des individus légèrement meilleurs en réalité avec une distance parcourue de 557 mm en moyenne ($\sigma = 489$ mm) et une disparité $S|R$ moyenne de 2.53 ($\sigma = 3.00$). Bien que la fitness en réalité augmente en moyenne de 146 mm par rapport à l'approche de référence ($\sigma = 90$ mm), ce qui est significativement plus grand que 0 d'après un test de Welch (p-value $= 3 \cdot 10^{-4}$), les contrôleurs trouvés se comportent toujours de manière médiocre en réalité. Une simple recherche locale ne permet donc pas de résoudre le problème du passage de la simulation à la réalité pour cette application.

On peut néanmoins émettre l'hypothèse que l'approche de référence ne trouve pas d'individus transférables, car elle ne trouve pas les solutions vraiment optimales en simulation et va sélectionner des contrôleurs sous-optimaux se transférant médiocrement sur le robot. L'approche de référence aidée d'un objectif de diversité comportementale est là pour tester cette hypothèse. L'introduction d'un objectif de diversité devrait en effet permettre d'améliorer l'exploration pendant l'optimisation et donc de trouver des solutions meilleures en simulation. C'est en effet le cas : les meilleures solutions trouvées correspondent à une distance parcourue moyenne de 1379 mm (85 mm de plus en moyenne que l'approche de contrôle simple) avec un écart-type de 54 mm. Néanmoins, comme indiqué sur la figure 4.18, ces contrôleurs se transfèrent encore moins bien sur le robot physique et obtiennent des performances moyennes de 385 en réalité ($\sigma = 279$ mm) avec une disparité $S|R$ moyenne de 3.20 ($\sigma = 3.22$). De plus, lors des 10 répétitions de l'approche, aucune des meilleures solutions trouvées en simulation n'est transférable en réalité. Contrairement à l'hypothèse avancée, l'approche de référence simple trouve donc parfois des individus performants et transférables, car elle sélectionne parfois des contrôleurs sous-optimaux. Ce résultat est consistant avec l'antagonisme entre les objectifs de performance et de transférabilité que nous avions supposé au début de cette section : en maximisant la fonction de fitness, on a tendance à s'écarter des zones transférables de l'espace de recherche.

Afin d'étudier plus en détail ce problème de passage de la simulation à la réalité dans le cadre de cette application de marche quadrupède, les meilleurs individus

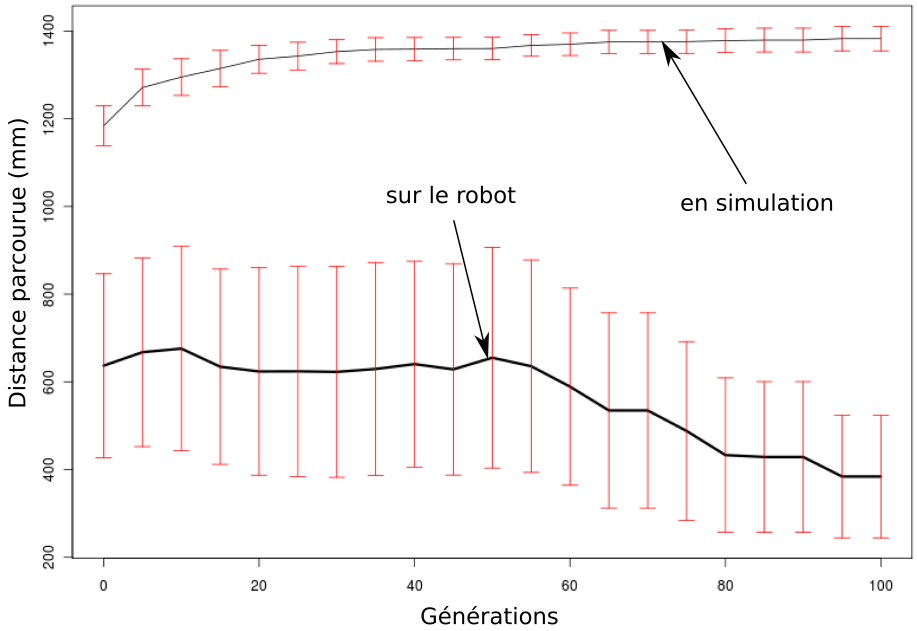

FIGURE 4.16 – Fitness des meilleures solutions obtenues sur les 10 répétitions de l'approche de référence & diversité comportementale toutes les 5 générations en simulation (courbe du haut) et en réalité (courbe du bas) : 210 expériences de transfert effectuées au total sur le robot. Les barres d'erreur indiquent une unité d'écart-type.

trouvés au cours de l'optimisation avec l'approche de référence & diversité comportementale ont été transférés sur le robot physique toutes les 5 générations. Les résultats obtenus sont représentés sur la figure 4.16, montrant pour les individus transférés leurs performances en simulation et en réalité (moyennes et écart-types). Ce graphique permet de mettre en valeur deux informations.

1. Les individus générés aléatoirement au début du processus ne sont ni transférables, ni performants en réalité.
2. Les solutions deviennent moins transférables et moins performantes sur le robot au cours de l'optimisation, signe d'un sur-apprentissage du modèle de simulation.

Ceci implique notamment qu'éviter le sur-apprentissage du simulateur, par exemple en préférant les solutions les plus robustes, ne serait pas suffisant ici pour trouver directement des solutions performantes en réalité.

4.4.3.2 Résultats sans simulation physique

Contrairement à l'application de navigation sur le robot e-puck, les optimisations dont l'évaluation se déroule uniquement en réalité permettent de trouver des solutions convenables, significativement plus performantes que celles obtenues avec les

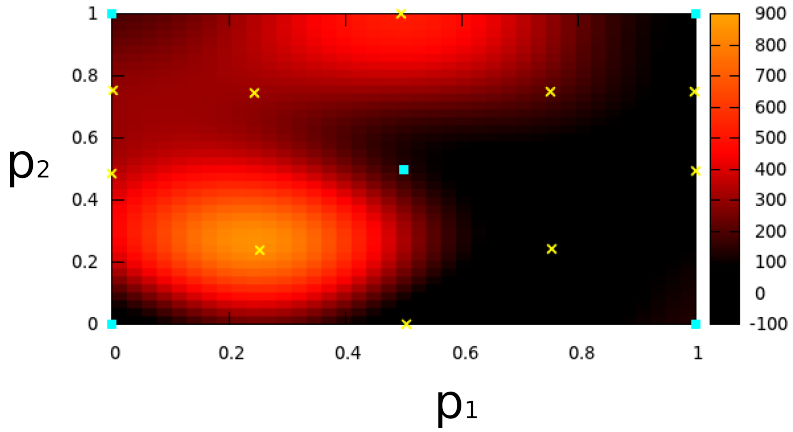

FIGURE 4.17 – Approximation par modèle de Kriging de la fitness en réalité construite lors de la meilleure optimisation avec l'approche utilisant un modèle de substitution par Kriging. Au total, 15 expériences ont été effectuées sur le robot en comptant les 5 expériences d'initialisation (indiquées par des carrés bleus). Les paramètres de contrôle correspondants aux différents transferts sont indiqués par les crois jaunes.

approches de référence. Il n'y a pas de différence significative entre ces 3 approches – évolution directe sur le robot, recours à un modèle de substitution de la fitness par IDW ou par Kriging – d'après un test de Welch (p-value > 0.73) avec des distances moyennes parcourues en réalité respectives de 693 mm (σ = 104 mm), 706 mm (σ = 79 mm) et 720 mm (σ = 108 mm). Ceci montre que le paysage de fitness en réalité est vraisemblablement plus simple comparé au paysage observé en simulation, étant donné que des processus d'optimisation avec peu d'évaluations fonctionnent relativement bien sur le robot. Nous reviendrons plus en détail sur ce point dans la partie 4.5.4.

4.4.3.3 Résultats avec l'approche par transférabilité

Le nombre de transferts moyen pendant l'optimisation avec l'approche par transférabilité sont disponibles dans la table 4.3. Le choix du seuil de diversité $\tau_{div} = 0.1$ permet environ 10 expériences de transfert. Néanmoins, même avec un faible nombre de transferts, l'approche par transférabilité permet de trouver des solutions significativement plus performantes et plus transférables que les 3 approches de référence d'après un test de Welch : les p-values sont inférieures à $6 \cdot 10^{-3}$ pour la distance parcourue et à $4 \cdot 10^{-2}$ pour la disparité $S|R$. Un des meilleurs individus compromis trouvé sur l'ensemble des 10 optimisations a une distance parcourue de 1132 mm en simulation et de 1099 mm sur le robot avec une valeur de disparité $S|R$ quasi-nulle de 0.004. En moyenne, l'approche trouve des solutions correspondant à une distance parcourue de 906 mm en simulation (σ = 210 mm) et 848 mm (σ = 239 mm) en réalité avec une disparité $S|R$ de 0.16 (σ = 0.25). Un comportement typique obtenu

Approches		Evaluation		Nombre d'expériences
		simulation	réalité	sur le robot physique
Evo. directe sur robot			x	20
Modèle de subst. de la fitness	IDW		x	10 (mean, sd = 1)
	Kriging		x	10
App. référence		x		1
App. référence & rech. loc.		x	x	10
App. référence & div.		x		1
App. par transf.		x	x	11 (mean, sd = 2)

TABLE 4.3 – Localisation de l'étape d'évaluation : totalement en simulation, totalement en réalité, ou dans les deux à la fois. Le nombre moyen d'expériences de transfert faites sur le robot est aussi indiqué pour chacune des approches.

avec l'approche par transférabilité est présenté sur la figure 4.20 (colonne de droite).

L'approche par transférabilité permet donc de trouver également pour cette application de marche quadrupède des solutions efficaces et qui se transfèrent bien sur le robot physique avec seulement une dizaine d'expériences de transfert pendant l'optimisation. Ce résultat renforce l'intérêt potentiel de l'approche pour résoudre le problème du passage à la réalité.

Comparé aux approches où l'évaluation des solutions se fait uniquement sur le robot, l'approche par transférabilité trouve de meilleures solutions en réalité, bien qu'il n'y ait pas de différence statistique très significative (p-values des tests de Welch < 0.1). Pour comprendre ce résultat, nous avons généré le paysage de fitness en réalité en fonction des deux paramètres de contrôle. Il s'avère que la fonction de fitness en réalité est uni-modale (voir figure 4.19), alors que la fonction de fitness en simulation est clairement multi-modale (voir figure 4.15). Ceci suggère, pour cette application, que la fonction de fitness en réalité est relativement simple, alors que la relation entre la performance simulée et la performance réelle semble beaucoup plus complexe. Pour des applications avec un plus grand nombre de paramètres de contrôle, des approches optimisant directement sur le robot risquent de nécessiter un grand nombre d'expériences pour obtenir de bons résultats

4.5 Expériences complémentaires

Suite à ces résultats sur deux applications robotiques, nous allons maintenant présenter une investigation rapide de points importants touchant à l'approche par transférabilité.

- Le modèle de substitution de la disparité $S|R$ interpolé par IDW est-il précis ?
- Comment définir efficacement la mesure de disparité $S|R$ et la métrique *in silico* ?

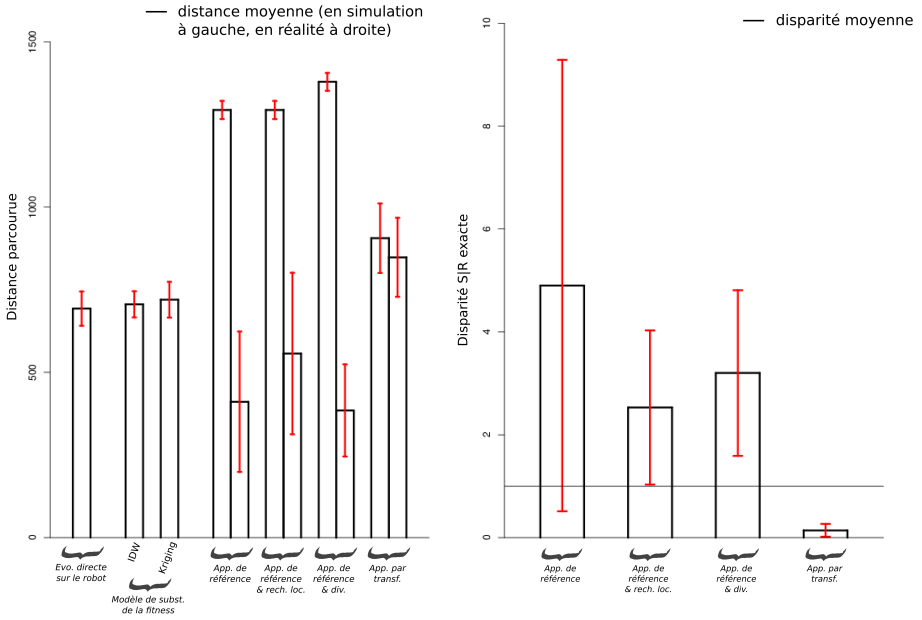

FIGURE 4.18 – Résultats obtenus dans l'application de marche quadrupède : distance parcourue en simulation et en réalité (mm, à gauche) et valeurs exactes de disparité (à droite) des meilleures solutions trouvées avec chaque approche. Moyennes et écarts types sont calculés sur 10 répétitions, sauf pour l'évolution directe sur le robot qui n'a été répétée que 3 fois. Les barres d'erreur indique une unité d'écart-type.

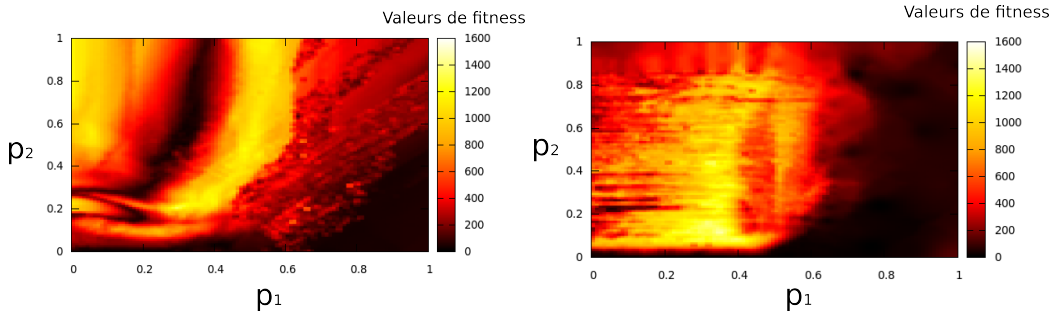

FIGURE 4.19 – A gauche, paysage de fitness (distance parcourue en millimètres) exhaustif obtenu en simulation pour l'application de marche quadrupède. A droite, interpolation du paysage de fitness en réalité basée sur environ 5500 expériences de transfert. Ces expériences ont essentiellement été menées dans la zone $p_1 < 0.6$: pour des valeurs plus élevées de p_1 le robot se recroqueville sur lui-même et ne bouge pas efficacement, d'où une région assez uniforme de fitness basses.

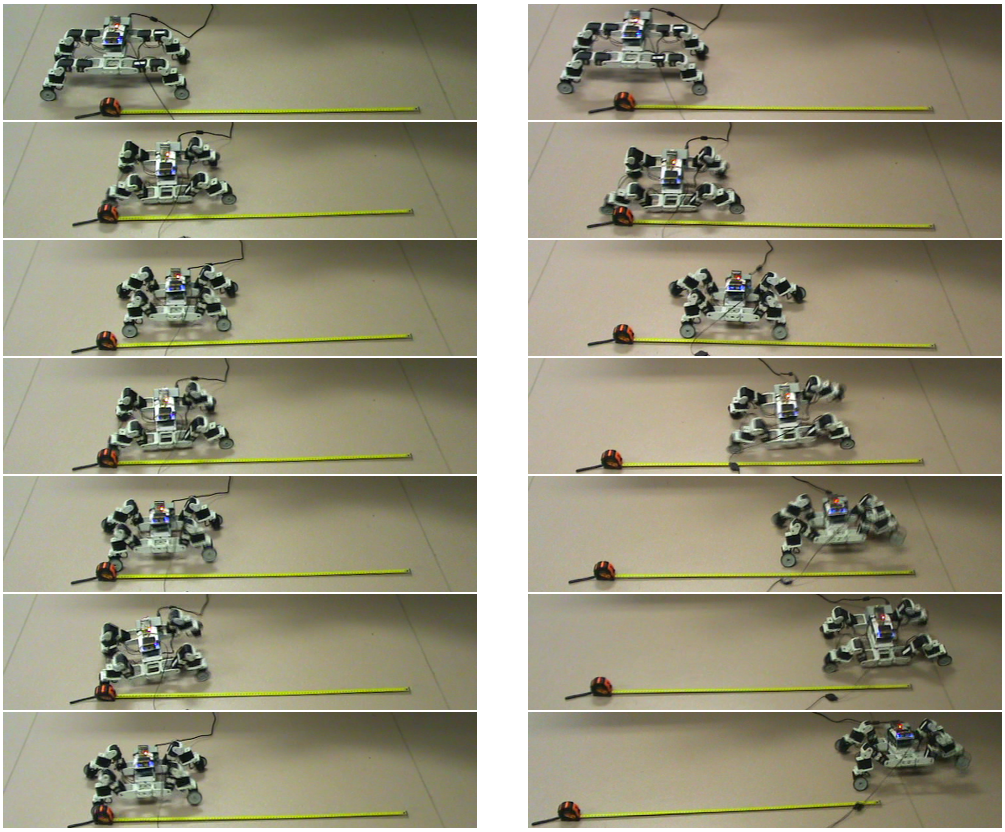

FIGURE 4.20 – A gauche, comportement typique évolué avec l'approche de référence : le comportement est rendu complètement inefficace à cause du glissement des pattes sur le sol. A droite, comportement typique obtenu avec l'approche par transférabilité : le robot effectue une marche quadrupède efficace et similaire au comportement observé en simulation.

- L'objectif de diversité et la stratégie de mise à jour du modèle de substitution sont-ils utiles ?

Afin de mener ces différentes investigations dans un temps acceptable, nous avons décidé de recourir à un dispositif expérimental purement simulé basé sur l'application de marche quadrupède. Au lieu d'un transfert de simulation à réalité, nous tentons de résoudre un problème de transfert entre une simulation simpliste et une simulation précise. La seule différence concerne la modélisation de l'asservissement des actuateurs du robot. Dans le simulateur simpliste, la vitesse d'un moteur est proportionnelle à l'erreur entre la position actuelle et la position désirée. Dans le simulateur précis, nous utilisons la véritable relation entre vitesse et erreur de position en nous basant sur la documentation des actuateurs Dynamixel AX-12+ [8]. Ce détail de simulation s'avère déterminant sur les comportements optimisés : les meilleures comportements dans le simulateur simple peuvent obtenir des fitness complètement irréalistes d'une dizaine de mètres parcourus en 10 secondes (14 mètres au maximum) alors que le simulateur précis correspond à des fitness maximales de 1400 mm environ. Les écarts de performance entre simulation et réalité seront donc potentiellement plus élevés que dans le dispositif expérimental précédent impliquant le robot physique.

4.5.1 Concernant le modèle de substitution

Pour évaluer si le modèle de substitution de la disparité $S|R$ est précis ou non, nous avons effectués 10 optimisations avec l'approche par transférabilité sur ce nouveau dispositif expérimental purement simulé. Le seuil de diversité est fixé à $\tau_{div} = 0.05$, correspondant à un nombre effectif moyen de transferts de 26 par optimisation ($\sigma = 6$). La figure 4.21 indique, pour l'ensemble des individus non-dominés de la dernière génération des différentes optimisations, les valeurs de disparité $S|R$ approchées selon le modèle de substitution en fonction des valeurs exactes de disparité $S|R$ correspondantes.

D'après ce graphique, le modèle de substitution tend à : (1) sur-estimer les valeurs pour les faibles valeurs de disparité $S|R$ exactes ; (2) sous-estimer la fonction exacte pour les grandes valeurs. Ces phénomènes sont essentiellement liés l'interpolation par Inverse Distance Weighting, car le modèle associé ne peut que prédire des valeurs dans le même intervalle que celui des points de données utilisés lors de sa construction.

Nonobstant cet effet de bord de l'interpolation par IDW, le coefficient de corrélation de Pearson entre les disparités $S|R$ approchée et exacte est relativement élevée : 0.76 en moyenne pour chaque optimisation ($\sigma = 0.11$). Des résultats similaires ont également été obtenus en utilisant une autre valeur pour le seuil de diversité $\tau_{div} = 0.025$, correspondant à un nombre moyen effectif d'expériences de transfert de 44 ($\sigma = 8$) : le coefficient de Pearson moyen est de 0.74 ($\sigma = 0.12$) sur 10 optimisations (résultats disponibles sur la figure 4.22). Il y a donc une forte

8. Ce simulateur précis correspondant en fait à la simulation utilisée dans les expériences précédentes avec le robot physique.

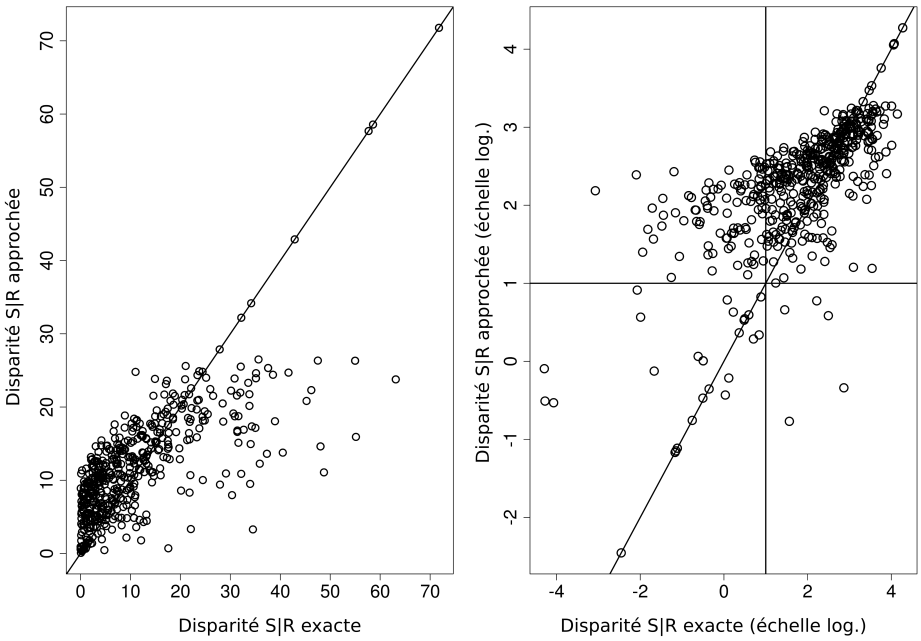

FIGURE 4.21 – Disparité $S|R$ approchées en fonction de la disparité $S|R$ exacte (en échelle logarithmique pour la figure droite) pour les solutions non-dominées à la fin de 10 optimisations effectuées avec l'approche par transférabilité (595 individus au total). Le seuil de diversité est $\tau_{div} = 0.05$, soit environ 25 transferts par optimisation. Les droites horizontale et verticale dans la figure de droite indiquent la valeur du seuil de disparité $D^*_{threshold} = 1$ utilisé.

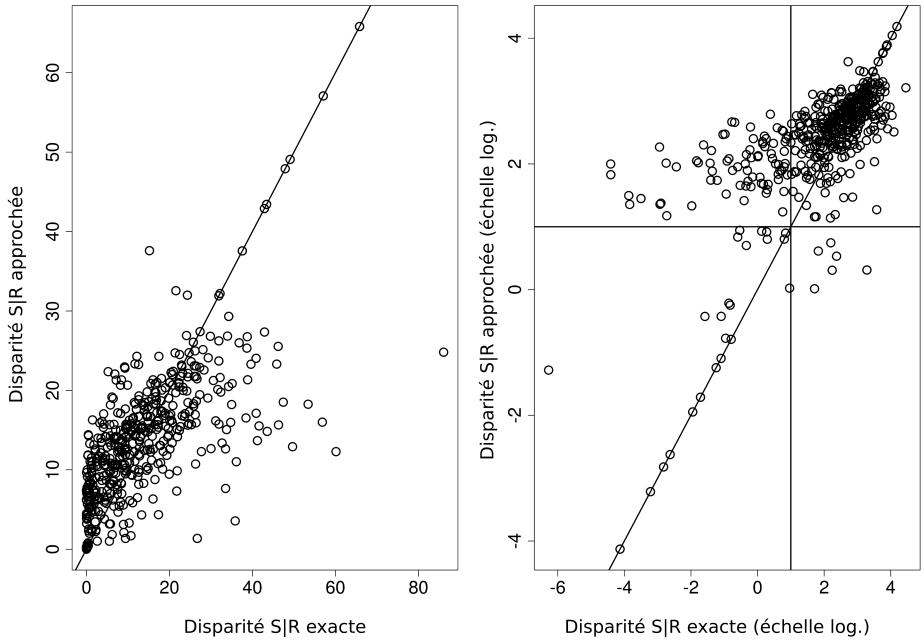

FIGURE 4.22 – Disparité $S|R$ approchée en fonction de la disparité $S|R$ exacte (en échelle logarithmique pour la figure droite) pour les solutions non-dominées à la fin de 10 optimisations effectuées avec l'approche par transférabilité (584 individus au total). Le seuil de diversité est $\tau_{div} = 0.025$, soit environ 45 transferts par optimisation. Les droites horizontal et verticale dans la figure de droite indiquent la valeur du seuil de disparité $D^*_{threshold} = 1$ utilisé.

relation de monotonie entre l'approximation et la fonction exacte. De tels résultats sont souvent suffisants pour conclure à la bonne qualité du modèle de substitution [Hüsken *et al.* 2005] : le modèle de substitution semble globalement fournir un gradient proche du gradient exact.

Néanmoins, dans notre approche, la fonction approchée de disparité $S|R$ n'est pas seulement un moyen de guider la recherche vers des zones transférables en simulation, mais aussi un critère numérique qui permet de distinguer les comportements transférables de ceux qui ne le sont pas. Il est donc également important de s'intéresser à l'erreur absolue entre les deux fonctions. En calculant l'erreur quadratique moyenne normalisée (nMSE) pour chaque optimisation, nous obtenons en moyenne une valeur de 0.43 avec $\tau_{div} = 0.05$ ($\sigma = 0.29$) et 0.38 avec $\tau_{div} = 0.025$ ($\sigma = 0.21$). Ces faibles valeurs de nMSE indiquent que la valeur de disparité $S|R$ approchée est globalement précise sur l'ensemble des solutions non-dominées, ce qui valide notre choix d'une simple interpolation par IDW pour construire le modèle de substitution, tout du moins pour cette application.

TABLE 4.4 – Première série de variantes : métrique *in silico* et disparité $S|R$.

Nom de la variante	Métrique *in silico* b_{dist}	Disparité $S\|R$ D^*
bCarac + DTraj	d_{carac}	d_{traj}
bCarac + DCarac	d_{carac}	d_{carac}
bTraj + DCarac	d_{traj}	d_{carac}
bTraj + DTraj	d_{traj}	d_{traj}

4.5.2 Concernant les distances comportementales : métrique *in silico* et disparité $S|R$

L'approche de transférabilité dépend de la définition de deux distances comportementales : (1) la mesure de disparité $S|R$ qui compare comportements simulés et réels ; (2) la métrique *in silico* qui compare les individus en simulation.

Dans l'application de marche quadrupède, nous avons défini deux distances comportementales. La première que nous appellerons d_{carac} est basée sur 3 caractéristiques comportementales que sont la distance parcourue, la hauteur moyenne du robot et son orientation horizontale finale. La seconde, d_{traj}, est directement basée sur la trajectoire des individus dans le plan horizontale. Suivant qu'on utilise l'une ou l'autre de ces distances en tant que mesure de disparité $S|R$ et en tant que métrique *in silico*, il est possible de définir 4 variantes comme indiqué dans la table 4.4. La variante *bCarac+DTraj* correspond à l'approche originale, utilisant les caractéristiques comportementales pour la métrique *in silico* et les trajectoires pour la mesure de disparité $S|R$.

Chacune de ces 4 variantes a été testée 10 fois dans le dispositif expérimental purement simulé avec deux valeurs de seuil de diversité : $\tau_{div} = 0.05$ (environ 25 transferts) et $\tau_{div} = 0.025$ (environ 45 transferts). Les résultats obtenus sont indiqués sur la figure 4.23. La variante *bCarac + DTraj* obtient les meilleurs résultats. La variante *bTraj + DTraj* utilisant uniquement la distance basée sur les trajectoires obtient des individus aussi transférables, mais globalement moins performants. Par contre, les deux variantes utilisant la distance basée sur les caractéristiques comportementales comme mesure de disparité $S|R$ (*bCarac + DCarac* et *bTraj + DCarac* obtiennent des résultats médiocres avec des individus non transférables. La distance basée sur les trajectoires semble donc plus efficace comme mesure de disparité $S|R$ qu'une distance plus simpliste basée sur des caractéristiques comportementales. Pour expliquer ce résultat, il est important de remarquer que la mesure de disparité $S|R$ n'est pas nécessairement une fonction de distance. Elle doit essentiellement assurer que de faibles valeurs de disparité $S|R$ correspondent à des individus transférables en définissant une série de contraintes sur la ressemblance entre le comportement simulé et celui observé sur le robot. De ce point de vue, la distance basée sur les caractéristiques comportementales imposent beaucoup moins de contraintes que celle utilisant les trajectoires. Pour que deux comportements soient proches d'après la distance d_{traj}, les trajectoires doivent en effet être à la fois proches dans l'espace,

mais également temporellement. Par contre, la distance d_{carac} ne juge que sur des caractéristiques moyennées ou instantanées, mais ne pose pas de contraintes sur tout le déroulement de l'expérience. Il est donc compréhensible que chercher à minimiser la distance d_{carac} ne soit pas suffisant pour trouver des comportements effectivement proches en simulation et en réalité, alors que minimiser la distance d_{traj} implique de fait une bonne correspondance spatio-temporelle entre les comportements.

Concernant maintenant la métrique *in silico*, on observe une différence significative entre les variantes *bCarac + DTraj* et *bTraj + DTraj*. Les deux variantes trouvent des individus transférables, mais avec des fitness significativement meilleures pour la première comparée à la seconde d'après un test de Welch (p-value $< 2 \cdot 10^{-4}$). Afin d'examiner ce point plus en détail, on compare la nMSE entre la disparité $S|R$ approchée par le modèle de substitution et la disparité $S|R$ exacte, toutes deux calculées pour les solutions non-dominées obtenues lors de chaque optimisation. Pour la variante *bCarac + DTraj*, on obtient une nMSE moyenne de 0.43 ($\sigma = 0.29$) avec 25 expériences de transferts et 0.38 ($\sigma = 0.21$) avec 45 expériences de transferts. Les valeurs sont beaucoup moins bonnes pour la variante *bTraj + DTraj* : en moyenne, des nMSE de 0.72 ($\sigma = 0.52$) et 1.13 ($\sigma = 0.79$) avec respectivement 25 et 45 transferts par optimisation. La prédiction du modèle de substitution est donc plus mauvaise si on utilise la trajectoire pour calculer la métrique *in silico* et augmenter le nombre d'expériences autorisé donne des modèles de substitution encore moins bons. Ces résultats indiquent que, pour cette application et les budgets d'évaluations proposés, la trajectoire dans le plan horizontal est un mauvais prédicteur de la valeur de disparité $S|R$, alors que les caractéristiques comportementales semblent plus appropriées. Nous avons en effet observé que pour certains contrôleurs, la trajectoire 2D ne permet pas de déterminer si le robot se déplace de manière réaliste ou non en simulation. Dans ce cas, deux comportements proches en terme de trajectoires vont se transférer différemment, ce qui implique d'effectuer plusieurs expériences ciblées dans cette zone de l'espace comportemental pour que le modèle de substitution puisse capter de telles variations. Avec le très petit budget d'évaluations sur le robot considéré, le modèle pourra difficilement être fiable pour ce genre de comportements. A l'opposé, les caractéristiques comportementales utilisées permettent de distinguer directement certains types de comportements qui risquent de se transférer différemment : les comportements qui sont efficaces ou non (distance parcourue), pour lesquels le robot tombe ou se retourne complètement (hauteur moyenne), qui glissent plus ou moins ou sont plus ou moins stables (orientation finale). L'utilisation de ces caractéristiques comportementales induit donc un partitionnement de l'espace a priori des comportements, ce qui peut faciliter la construction du modèle de substitution.

Ces résultats semblent donc valider notre approche originale avec une distance comportementale basée sur des caractéristiques comme métrique *in silico* et une distance basée sur les trajectoires comme mesure de disparité $S|R$.

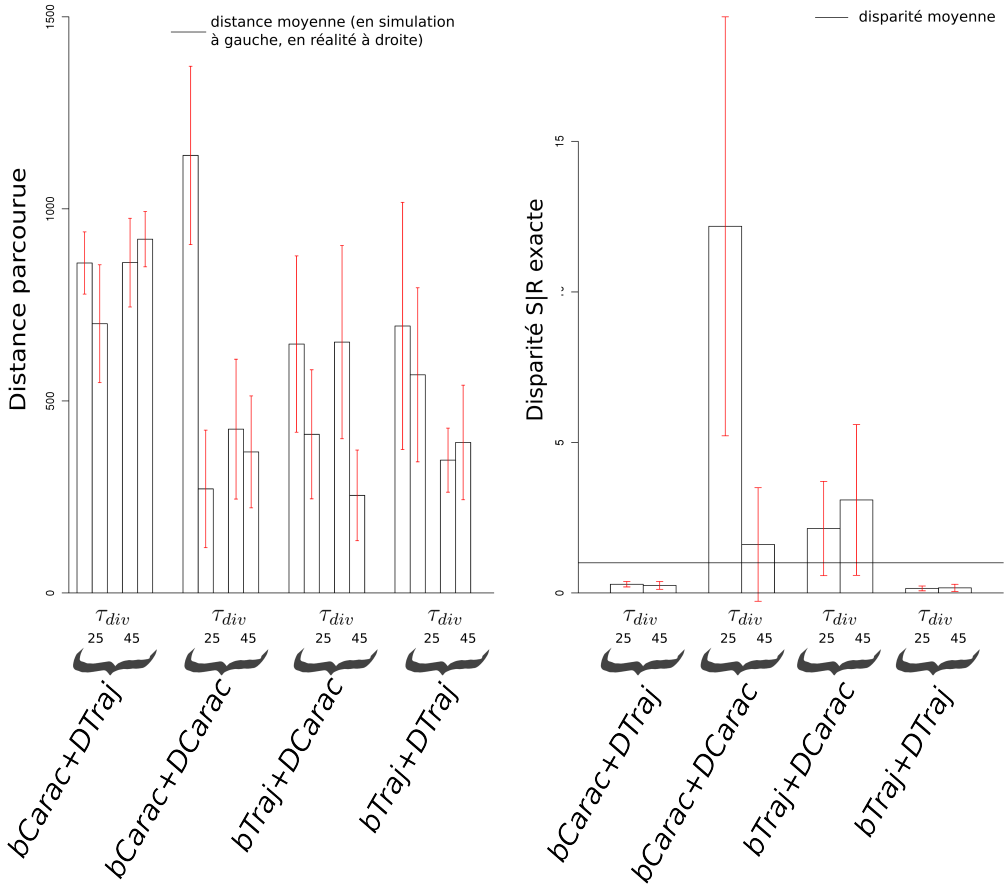

FIGURE 4.23 – Comparaisons entre 4 variantes utilisant différentes distances comportementales en tant que mesure de disparité $S|R$ et métrique *in silico* : distance parcourue (mm, figure de gauche) dans les simulations simple et précise et valeurs de disparité $S|R$ (figure de droite) des meilleures solutions obtenues dans 10 répétitions avec chaque variante. Le nombre moyen approximatif d'expériences de transferts par optimisation est indiqué au-dessus du nom de chaque variante. Les barres d'erreur indiquent une unité d'écart-type.

TABLE 4.5 – Seconde série de variantes : stratégie de mise à jour et objectif de diversité.

Variantes	Objectif de diversité	Stratégie de mise à jour
RandomT & Div	×	random
MaxDivT & Div	×	max. diversité
RandomT & NoDiv		random

4.5.3 Concernant la stratégie de mise à jour et l'objectif de diversité

Nous avons ensuite implémenté une deuxième série de variantes qui fait varier : (1) la stratégie de mise à jour du modèle de substitution ; (2) la présence/absence d'un objectif de diversité. Nous introduisons deux stratégies possibles :

- la stratégie *random* consiste à choisir aléatoirement le contrôleur à transférer parmi ceux dont la diversité dépasse un certain seuil τ_{div} ;
- la stratégie *max. diversité* revient à transférer systématiquement le contrôleur dont la diversité est maximale, à condition que sa valeur de diversité dépasse le seuil τ_{div}.

La table 4.5 présente les différentes variantes définies. La variante *RandomT & Div* utilisant la stratégie *random* et un objectif de diversité correspond à l'approche originale. Rappelons que l'objectif de diversité utilisé est la distance minimale, d'après la métrique *in silico*, du contrôleur considéré aux contrôleurs ayant déjà été transférés.

Chaque variante a été répétée 10 fois sur le dispositif expérimental purement simulé et testée avec deux valeurs de seuil de diversité : $\tau_{div} = 0.05$ pour 25 transferts et $\tau_{div} = 0.025$ pour 45 transferts en moyenne. Les résultats sont indiqués sur le graphique 4.24. Les variantes utilisant un objectif de diversité obtiennent les meilleurs résultats avec des solutions performantes et toujours transférables (valeurs de disparité inférieures à $D^*_{threshold}$). La variante *RandomT & Div* correspond à des individus significativement meilleurs que *MaxDivT & Div* en terme de disparité $S|R$ avec $\tau_{div} = 0.05$ (p-value = 0.071 avec un test de Welch) et en terme de distance parcourue avec $\tau_{div} = 0.025$ (p-value = 0.029 avec un test de Welch). Ceci signifie donc que transférer systématiquement l'individu le plus éloigné des contrôleurs déjà transférés, donc potentiellement plus informatif, ne permet pas d'améliorer le modèle de substitution par rapport à la stratégie *random*. Une explication envisageable est que le point dans l'espace des caractéristiques comportementales le plus éloigné des contrôleurs déjà transférés serait, au moins pour les premiers transferts, souvent situé sur les bords de l'espace comportemental. Il correspondrait donc à des comportements extrêmes ou peu typiques dont l'information de transférabilité ne serait pas très informative, car valable pour une classe très spécifique et restreinte de comportements. Si cette hypothèse est vraie, la stratégie *random* permettrait de construire un modèle de substitution plus fiable pour un budget d'évaluations

donné, car pouvant sélectionner des individus plus "intermédiaires" dans l'espace de comportement.

Les mauvais résultats de la variante *RandomT & NoDiv*, notamment en terme de disparité $S|R$, indiquent clairement que la stratégie *random* n'est pas suffisante seule pour permettre une mise à jour efficace du modèle de substitution : une pression explicite vers de nouvelles zones intéressantes de l'espace de recherche est nécessaire, ici à l'aide de l'objectif de diversité comportementale.

4.5.4 Exploitation des paysages de fitness

La génération des paysages de fitness quasi-exhaustifs en simulation et en réalité nous permet de répondre à certaines questions dans le cadre de cette application à la marche quadrupède.

Pourquoi l'optimisation en simulation n'est-elle pas fructueuse ? Comme indiqué sur la figure 4.25, la simulation comporte deux zones de l'espace de recherche avec de hautes fitness, donc susceptibles d'être sélectionnées lors d'une optimisation en simulation. La zone 1 correspond à des individus dont le paramètre de contrôle p_2 réglant l'amplitude des mouvements est élevé. Les contrôleurs de cette zone sont performants en simulation, du fait de leur dynamique très instable, mais l'équivalent en réalité donne toujours de mauvaises valeurs de fitness. Par contre, concernant la zone 2, certains comportements vont être aussi performants en réalité et d'autres non. Ceci explique pourquoi dans cette application, certaines optimisations en simulation (3 sur 10 répétitions) donnent des individus performants en réalité. Mais, comme il n'y a aucune façon de déterminer quels contrôleurs vont bien se comporter sur le robot physique d'après la seule fitness en simulation, il n'y a aucune garantie que le meilleur contrôleur en simulation sera effectivement intéressant en réalité.

Pourquoi l'optimisation directe sur le robot physique ne donne-t-elle pas de meilleurs résultats ? Si on s'intéresse maintenant uniquement au paysage de fitness en réalité, on se rend compte qu'il est relativement simple, avec une unique zone de haute fitness. Néanmoins, les différents essais d'optimisation directe en réalité n'ont permis que de découvrir des contrôleurs de performances moyennes. L'explication en est relativement simple : le nombre d'expériences autorisées par optimisation est trop faible pour permettre d'explorer efficacement l'espace de recherche. Par exemple, si on considère les optimisations effectuées à l'aide d'un modèle de subsitution par Kriging, l'approximation de la fitness obtenue après les 15 expériences est très éloignée du paysage de fitness réel (voir figure 4.26). Il est tout à fait probable qu'en doublant le nombre d'expériences de transfert autorisées, cette zone optimale aurait été découverte. Il est cependant important de noter que, bien que cette application donne lieu à un problème de passage de simulation à réalité non trivial, l'optimisation de seulement deux paramètres la rend relativement triviale si on ne considère que la fonction de fitness réelle. Dans des cas plus réalistes, optimiser sur le robot risque de nécessiter un nombre prohibitif d'expériences.

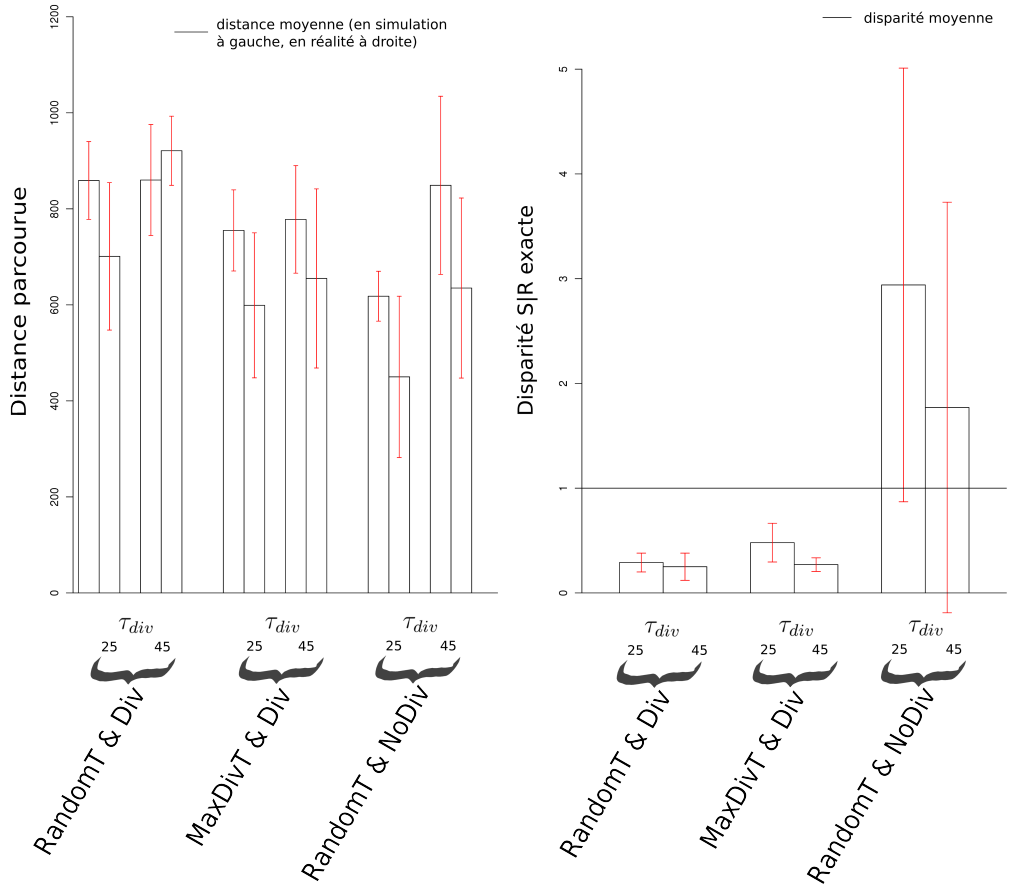

FIGURE 4.24 – Comparaisons entre 2 stratégies de mise à jour du modèle de substitution de la disparité $S|R$ et évaluation de l'influence de l'objectif de diversité : distance parcourue (mm, figure de gauche) dans les simulations simple et précise et valeurs de disparité $S|R$ (figure de droite) des meilleures solutions obtenues dans 10 répétitions avec chaque variante. Le nombre moyen approximatif d'expériences de transferts par optimisation est indiqué au-dessus du nom de chaque variante. Les barres d'erreur indiquent une unité d'écart-type.

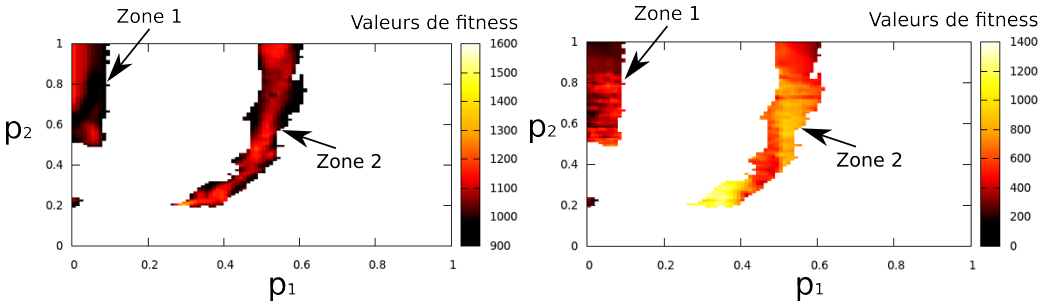

FIGURE 4.25 – A gauche, zones où la fitness est supérieure à 900 mm en simulation. A droite, fitness observées en réalité dans les zones correspondantes. Les échelles de couleur sont différentes d'une figure à l'autre.

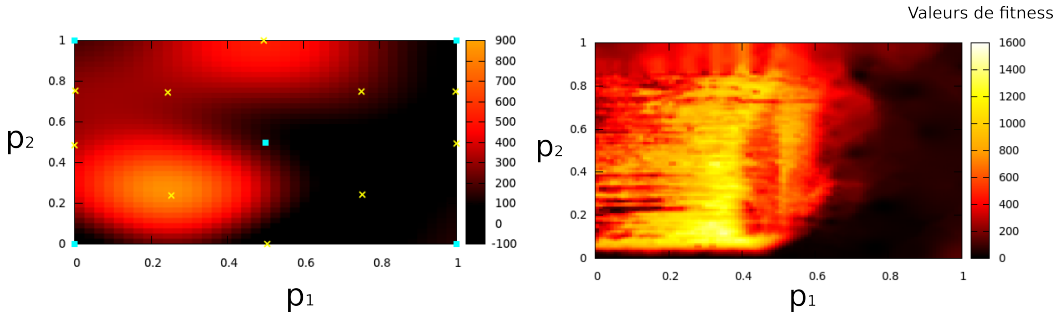

FIGURE 4.26 – A gauche, paysage de fitness typique obtenu lors des optimisations à partir d'un modèle de subsitution utilisant le Kriging. A droite, paysage de fitness en réalité.

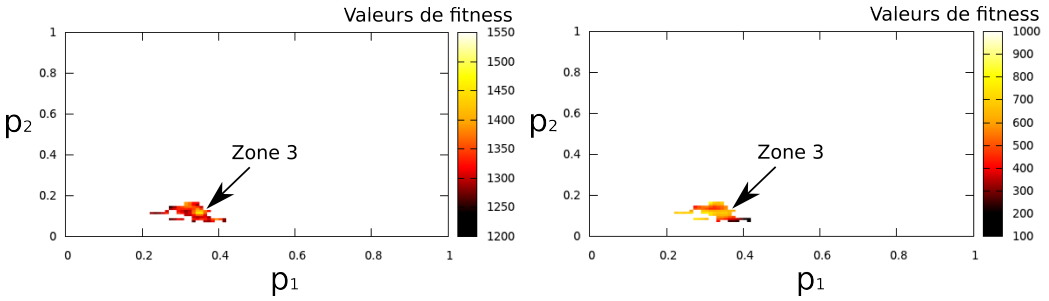

FIGURE 4.27 – A gauche, zone des meilleures fitness en réalité. A droite, fitness de la zone correspondante en simulation.

L'approche par transférabilité permet-elle de trouver les individus optimaux en réalité ? D'après les résultats, même si l'approche par transférabilité permet d'obtenir des individus significativement meilleurs que d'autres méthodes plus classiques, elle ne permet pas de trouver la zone de fitness optimale en réalité. En effet pour cette application à la marche quadrupède, la zone de fitness maximale en réalité correspond à une zone de basse fitness en simulation. D'après la figure 4.27, les fitness réelles dans cette zone vont de 1200 à 1600, alors que les fitness correspondantes en simulation sont toujours inférieures à 1000. Ainsi les comportements optimaux en réalité ne peuvent pas être obtenus si on cherche des comportements simulés transférables : la contrainte de transférabilité nous rend ainsi dépendant de la capacité du simulateur à reproduire les comportements les plus intéressants en réalité.

Quelles zones de l'espace de recherche sont privilégiées par l'approche par transférabilité ? En regardant en détail les individus trouvés par l'approche par transférabilité, on s'aperçoit qu'ils sont regroupés dans une zone précise de l'espace de recherche, comme indiqué sur la figure 4.28. Cette zone a la caractéristique de correspondre à des individus similairement performants en simulation et en réalité, avec des vitesses de marche sur le robot physique d'environ 1 mètre en 10 secondes. Ces résultats indiquent, pour cette application à la marche quadrupède, qu'optimiser la transférabilité telle que nous l'avons définie permet effectivement d'obtenir une correspondance entre les performances en simulation et en réalité.

4.6 Discussion

4.6.1 Antagonisme entre performance et transférabilité

L'approche par transférabilité propose d'optimiser les contrôleurs suivant trois objectifs, dont la fitness en simulation et une valeur de disparité approchée calculée à l'aide d'un modèle de substitution. Comme nous avons fait l'hypothèse que ces deux objectifs sont antagonistes, nous avons proposé d'évaluer les individus dans un

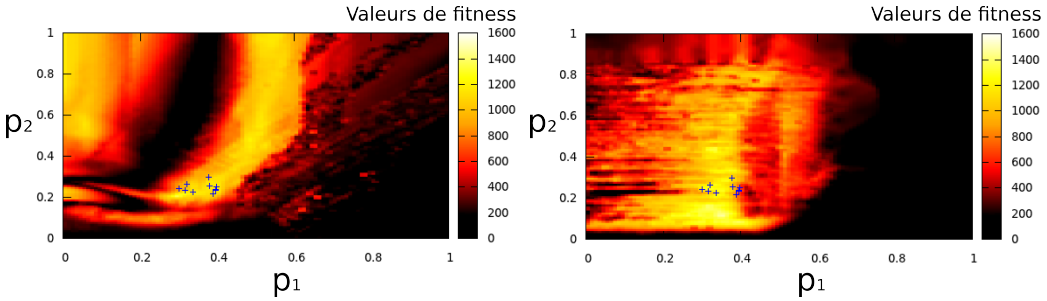

FIGURE 4.28 – Localisation des solutions trouvées lors des 10 optimisations effectuées avec l'approche par transférabilité dans les paysages de fitness : à gauche en simulation, à droite en réalité.

schéma d'optimisation multi-objectif. L'existence de cet antagonisme peut maintenant être discuté à la lumière des résultats obtenus sur les 2 applications robotiques.

Un premier résultat indiquant la présence d'un antagonisme dans notre application de marche quadrupède est la différence de performances observée entre l'approche de référence et l'approche de référence & diversité : alors que l'ajout d'un objectif de diversité permet de trouver de meilleurs individus en simulation, les performances obtenues après transfert sur le robot physique sont moins bonnes. Ceci indique que les solutions les plus efficaces trouvées en simulation ne sont pas transférables et donc que la fonction de fitness va sélectionner des individus se révélant peu efficaces en réalité. Ainsi, optimiser directement cette fonction de fitness ne donnera aucune garantie quant à la qualité finale de la solution trouvée sur le robot et donc sur sa transférabilité. Ceci semble vérifier empiriquement notre hypothèse d'antagonisme entre performance et transférabilité de simulation vers réalité.

Afin de comprendre plus en détail les liens entre performances et disparité $S|R$, nous avons également généré les paysages de fitness en simulation et en réalité pour cette application dans l'espace des deux paramètres de contrôle. Le nombre de contrôleurs possibles étant tout de même élevé (exactement $2.6 \cdot 10^5$), le paysage en réalité n'est pas exhaustif et a été construit à l'aide d'environ 5500 expériences sur le robot. En pratique, le paysage est quasiment exhaustif, car les zones où peu d'expériences ont été effectuées correspondent aux jeux de paramètres qui ne permettent pas au robot de marcher convenablement (notamment dans le cas le paramètre p_1 est supérieur à 0.6). Pour ces zones, les valeurs de la fonction ont été interpolées pour obtenir le paysage complet. Le paysage de fitness en simulation a lui été construit en effectuant environ 10^4 évaluations régulièrement distribuées dans l'espace des paramètres. Les figures des paysages de fitness sont visibles sur la figure 4.19 pour les paysages de fitness. En s'intéressant aux zones de la simulation plutôt performantes indiquées sur la figure 4.25, il s'avère qu'elles correspondent majoritairement à des zones de performances moyennes ou faibles. Les contrôleurs concernés subissent une perte importante de performances quand ils sont transférés sur le robot : ils ne sont pas transférables. Ce résultat prouve l'existence d'un antagonisme entre la performance en simulation et la performance en réalité, et donc entre la performance en

simulation et l'objectif de transférabilité.

4.6.2 Vers une mesure de transférabilité embarquée

L'estimation de la mesure de transférabilité implique d'avoir à disposition le comportement réel du robot correspondant au contrôleur évolué. Dans les expériences présentées ici, la trajectoire du robot utilisée pour calculer la disparité $S|R$ exacte est obtenue de manière externe à l'aide d'un processus de capture de mouvement utilisant des scanners cx1 CODA, ce qui n'est pas toujours envisageable pour des robots plus volumineux ou dans des environnements extérieurs. Un problème important concernant l'applicabilité de l'approche va donc être la possibilité de définir une mesure de disparité $S|R$ fiable à partir d'un comportement interne. Nous traiterons ce point plus en détail à la section 6.4.1 dans la dernière partie expérimentale du manuscrit portant sur l'adaptation de robots à leur environnement.

4.6.3 Modéliser la fitness ou la transférabilité

L'utilisation des modèles de substitution est croissante en robotique, le plus souvent afin de construire directement une approximation de la fonction de performance sur le robot physique. Ces modèles prennent habituellement en entrée l'espace des paramètres du contrôleur. L'hypothèse implicite de la plupart des modèles de substitution est que la proximité dans l'espace d'entrée va signifier une proximité dans la valeur de sortie. Autrement dit ici, des vecteurs de paramètres proches (i.e. des génotypes proches) devraient correspondre à des valeurs de fitness proches. Cette hypothèse est difficilement vérifiable, voire très vraisemblablement fausse avec certaines structures de contrôle : par exemple dans un réseau de neurones, la modification du poids d'une seule connexion peut changer complètement sa dynamique globale. Dans ces conditions, on peut donc se demander si le génotype est un prédicteur efficace de la fitness.

D'autre part, modéliser directement la fitness à partir du génotype signifie que l'on n'utilise aucune information *a priori* sur le robot et le paysage de fitness, contrairement aux travaux se basant sur une simulation. Il est alors souvent nécessaire de commencer avec un contrôleur non trivial de fitness non nulle [Hemker *et al.* 2006] pour éviter des problèmes de bootstrap et limiter le nombre d'expériences dans des régions de l'espace de recherche peu intéressantes. Les résultats obtenus avec un modèle de substitution basé sur le génotype dans les deux applications robotiques considérées sont très dépendants du paysage de fitness en réalité et de la taille de l'espace de recherche : pour l'application de marche quadrupède où il n'y a que deux paramètres ($2.6 \cdot 10^5$ contrôleurs possibles), ces méthodes trouvent des solutions correctes bien que loin de l'optimum (voir figure 4.19). Pour l'application de navigation d'un robot e-puck qui compte 35 paramètres ($1.4 \cdot 10^{42}$ contrôleurs possibles), il faudrait vraisemblablement beaucoup plus qu'une dizaine d'expériences pour trouver des solutions qui commencent à résoudre la tâche. Dans le cas d'applications robotiques réalistes, on peut donc imaginer que de nombreuses expériences vont être

nécessaires pour construire, à partir du génotype, un modèle de substitution précis sans information *a priori* sur le robot.

Si l'on cherche au maximum à réduire le nombre d'expériences sur le robot, nos résultats expérimentaux suggèrent qu'il serait plus efficace de construire un modèle de substitution à partir d'une description comportementale issue d'un simulateur, plutôt qu'à partir du génotype. Néanmoins, on peut dans ce cas se demander s'il vaut mieux construire et optimiser directement une approximation de la fitness réelle à partir du comportement simulé (cas 1) ou optimiser à la fois la fitness simulée et un objectif de transférabilité approximée comme le propose l'approche par transférabilité (cas 2) ? L'avantage de la première technique est qu'obtenir un bon modèle de la fitness réelle permet de trouver les comportements optimaux en réalité. L'approche par transférabilité va par contre restreindre la recherche dans les zones transférables et peut donc s'éloigner de l'optimum réel si celui-ci est considéré comme non transférable. Dans le cas 1, tout le processus d'optimisation est néanmoins dépendant du modèle de la fitness réelle, aussi bien pour comparer les solutions que pour générer les nouvelles expériences à effectuer sur le robot. On peut donc difficilement garantir que les expériences effectuées au début de l'optimisation seront informatives. Dans le cas 2, optimiser la fitness en simulation permet de biaiser la recherche vers des contrôleurs potentiellement intéressants et le choix des expériences à mener en réalité peut prendre en compte ce critère. A mon sens, les deux approches sont viables, mais l'approche de transférabilité devrait nécessiter moins d'expériences sur le robot dans le cas où le simulateur est de bonne qualité.

4.6.4 Mettre à jour la simulation à l'aide de la disparité $S|R$

L'approche par transférabilité optimise les solutions dans un simulateur fixe et n'a pas pour but d'améliorer le modèle de simulation. Parmi les perspectives envisageables, il est néanmoins intéressant de remarquer que le modèle de substitution de la disparité $S|R$ obtenu durant une optimisation permet de distinguer, plus ou moins précisément, les comportements bien modélisés en simulation des comportements non transférables. A partir de cette information, il est possible de déterminer quels phénomènes physiques modélisés par le simulateur interviennent dans les comportements non transférables et, à terme, d'isoler ceux qui rendent la simulation peu fiable pour cet ensemble de comportements. L'approximation de la disparité $S|R$ peut alors être interprétée comme une fonction de qualité du simulateur bâtie sur un espace comportemental et pourrait être utilisée pour comprendre quelles parties du modèle de simulation ne se comportent pas comme dans la réalité. Il serait alors envisageable d'utiliser cette connaissance pour améliorer la simulation en interaction avec des experts en robotique et en mécanique. Ce point sera discuté plus en détail dans la partie 7.3 de la discussion.

4.7 Conclusion

Ce chapitre traite du problème du passage de la simulation à la réalité qui limite nombre d'applications de robotique évolutionniste à la simulation. Afin de s'attaquer à cette question de manière générale, nous proposons l'approche par transférabilité qui vise à trouver des contrôleurs à la fois performants en simulation et transférables sur le robot physique. Il s'avère que cette approche donne de bons résultats sur deux applications robotiques, la première de navigation dans un labyrinthe simple et la seconde visant à optimiser des comportements de marche quadrupède. L'approche par transférabilité semble également plus performante que les méthodes classiquement utilisées pour l'optimisation de contrôleurs, notamment la méthodologie des simulations minimales de Jakobi.

Chapitre 5

Doter les contrôleurs de capacités de généralisation

La plupart des travaux présentés dans ce chapitre ont été effectués en collaboration avec Tony Pinville de l'Institut des Systèmes Intelligents et de Robotique de Paris et ont fait l'objet d'une publication à la conférence *GECCO* en 2011 dans l'article [Pinville *et al.* 2011].

5.1 Problématique et approche

Optimiser des contrôleurs en robotique évolutionniste revient classiquement à évaluer chacune des solutions considérées dans quelques contextes [Jakobi 1997, Jakobi 1998a, Berlanga *et al.* 2002, Ziemke & Thieme 2002, Barate & Manzanera 2008, Doncieux & Mouret 2010, Mouret & Doncieux 2011], voire dans un unique contexte [Reynolds 1994, Miglino *et al.* 1995, Jakobi *et al.* 1995, Kondo *et al.* 1999, Floreano & Mondada 1998, Floreano & Urzelai 2001, Urzelai & Floreano 2001]. Le meilleur contrôleur est donc sélectionné sur des comportements spécifiques à quelques contextes précis et il n'y donc aucune garantie qu'il se comporte aussi efficacement sur des contextes différents [1], qu'ils soient proches ou non. Or, dans de nombreuses applications robotiques, le robot doit gérer de multiples situations sur lesquelles il n'a pas forcément été testé lors de l'optimisation : le robot doit donc faire preuve de capacités de généralisation afin d'assurer une performance élevée dans toutes les situations rencontrées. Prenons par exemple le cas d'un robot devant se déplacer dans une pièce : dans l'idéal, sa performance de navigation ne doit pas dépendre de la taille de la pièce, de sa position initiale ou encore de l'agencement précis des obstacles. Pour ce faire, le robot peut s'adapter en ligne à son environnement [Kondo *et al.* 1999, Floreano & Urzelai 2001, Urzelai & Floreano 2001], par exemple en modifiant son propre contrôleur pendant qu'il résout la tâche. Dans cette section, nous nous intéresserons à l'autre alternative : générer des contrôleurs par évolution dotés de capacités de généralisation.

Comme évaluer dans un unique contexte ne permet pas de trouver des comportements capables de généraliser efficacement, pourquoi ne pas évaluer les contrôleurs sur un ensemble plus représentatif de contextes ? Dans un processus évolutionniste classique où chaque contrôleur est évalué sur chacun des contextes,

1. Un contexte peut souvent être décrit comme un vecteur de valeurs spécifiques attribuées à chaque paramètre de la tâche et nous nous limiterons à cette définition dans ce chapitre.

il est évident qu'utiliser un plus grand nombre de contextes va augmenter sensiblement le nombre total d'évaluations : le processus d'optimisation peut en être considérablement ralenti. En pratique, les travaux en robotique évolutionniste qui visent à améliorer les capacités de généralisation des contrôleurs en augmentant directement le nombre de contextes d'évaluation se limitent souvent à moins de 10 contextes [Jakobi 1998a, Berlanga *et al.* 2002, Ziemke & Thieme 2002, Barate & Manzanera 2008, Doncieux & Mouret 2010]. Bien qu'une dizaine de contextes puissent suffire dans les applications présentées dans ces travaux, une telle méthode ne peut pas être utilisée de manière générique si on souhaite maintenant généraliser sur plusieurs centaines de contextes. La question de doter les contrôleurs de capacités de généralisation est donc directement reliée au besoin de limiter le nombre d'évaluations pendant l'optimisation.

Une autre limitation concernant la plupart des travaux de la littérature est que la capacité de généralisation des contrôleurs obtenus est rarement évaluée. Les solutions sont uniquement testées dans les contextes sur lesquels elles ont été optimisées [Jakobi 1998a, Berlanga *et al.* 2002, Ziemke & Thieme 2002]. Les contrôleurs sélectionnés ont donc des capacités de généralisation apparemment bonnes. Néanmoins, comme ils ont été directement optimisés sur les contextes de tests, on ne peut rien affirmer concernant leurs capacités de généralisation effectives sur de nouveaux contextes.

Ce problème de généralisation peut être interprété comme un cas typique de sur-apprentissage. Un contrôleur qui maximise la performance sur l'ensemble des contextes d'apprentissage risque d'exploiter des spécifités de ces contextes : il pourra ne pas être du tout performant dans des contextes différents où ces spécificités n'apparaissent pas. Autrement dit, une performance optimale sur l'ensemble d'apprentissage ne donnera aucune garantie quant à la capacité de généralisation du contrôleur obtenu. L'apprentissage supervisé propose une méthodologie [Alpaydin 2004, Gagné & Schoenauer 2006] visant à la fois à détecter les effets du sur-apprentissage tout en permettant d'évaluer efficacement la capacité de généralisation. Cette méthodologie présentée dans la partie 2.4.2.1 repose sur la définition de trois ensembles de contextes :

- un ensemble de contextes d'apprentissage Ω_{train} utilisé pour optimiser les solutions ;
- un ensemble de contextes de validation Ω_{valid} permettant d'estimer la capacité de généralisation de solutions prometteuses sur Ω_{train} ;
- un ensemble de contextes de test Ω_{test} évaluant, une fois l'optimisation terminée, la capacité de généralisation des meilleures solutions trouvées sur de nouveaux contextes.

La capacité de généralisation d'un contrôleur c sur un ensemble de contextes Ω est la somme des performances de c sur les différents contextes de Ω. Afin de permettre une estimation non biaisée de cette capacité de généralisation une fois l'optimisation terminée, il est important que les ensembles de validation et de test soient disjoints : $\Omega_{valid} \cap \Omega_{test} = \emptyset$ [Alpaydin 2004]. Comme nous l'avons indiqué précédemment (cf. partie 2.4.2), une telle distinction est rarement faite dans les

travaux de robotique évolutionniste [Barate & Manzanera 2008].

L'approche que nous allons maintenant proposer se base sur une méthodologie inspirée de celle de l'apprentissage supervisée. De la même manière, trois ensembles de contextes - d'apprentissage, de validation et de test - vont être définis. Néanmoins, contrairement aux travaux d'apprentissage supervisé qui suggèrent d'utiliser un ensemble d'apprentissage de plus grand taille que l'ensemble de validation [Drucker *et al.* 1997, Kohavi 1995, Alpaydin 2004], nous imposons ici que l'ensemble d'apprentissage soit de petite taille afin de limiter le nombre d'évaluations effectuées au cours du processus d'optimisation.

5.1.1 L'approche *ProGAb*

Chercher à améliorer les capacités de généralisation des contrôleurs met en lumière le conflit suivant : (1) alors que la valeur de performance sur un petit ensemble d'apprentissage Ω_{train} est rapide à calculer, elle guide l'optimisation vers des solutions qui sur-apprennent et donc avec de faibles capacités de généralisation ; (2) par contre, même si évaluer les solutions sur un ensemble de validation Ω_{valid} de grande taille est plus informatif, ce ne peut être fait systématiquement à cause du coût en temps de calcul trop important.

Notre approche est inspirée de l'approche par transférabilité présentée dans le chapitre précédent et propose donc un schéma d'optimisation similaire. Partant du constant qu'être performant sur l'ensemble d'apprentissage Ω_{train} ne garantit pas d'être performant sur l'ensemble de validation Ω_{valid}, l'approche *ProGAb* ("Promoting the Generalisation Ability") se propose d'optimiser les solutions via un algorithme évolutionniste multi-objectif où chaque contrôleur est évalué par les objectifs suivants : (1) la performance F sur l'ensemble d'apprentissage Ω_{train} ; (2) la capacité de généralisation G^* calculée sur l'ensemble de validation Ω_{valid} ; (3) un objectif de diversité comportementale pour explorer plus efficacement l'espace de recherche [Trujillo *et al.* 2008, Gomez 2009, Mouret & Doncieux 2011] (cf. partie 2.3.3).

De plus, comme évaluer chaque contrôleur sur l'ensemble Ω_{valid} serait trop coûteux en temps de calcul, nous proposons que la capacité de généralisation sur Ω_{valid} soit approximée à l'aide d'un modèle de substitution construit et mis à jour pendant le processus d'optimisation en évaluant seulement quelques contrôleurs sur l'ensemble Ω_{valid}.

La méthode ProGAb peut être interprétée comme une technique de multi-objectivisation (voir section 2.3.1). Le but de l'optimisation est en effet de maximiser la performance sur l'ensemble d'évaluation ($\Omega_{eval} = \Omega_{train} \cup \Omega_{valid}$), et la méthode introduit deux objectifs séparés : la performance d'apprentissage sur Ω_{train} et la performance de validation sur Ω_{valid}.

5.1.1.1 Modèle de substitution de la capacité de généralisation

Les modèles de substitution sont classiquement utilisés dans des problèmes d'ingénierie où évaluer une solution est trop long/coûteux : une approximation de la fonction de performance est alors construite pendant l'optimisation et, au lieu d'évaluer les solutions directement sur le système physique testé, elles sont optimisées suivant ce modèle approché de la performance. Nous sommes typiquement dans un cas similaire, le calcul de la capacité de généralisation G^* de chaque contrôleur sur l'ensemble de validation Ω_{valid} impliquant un très grand nombre d'évaluations. Ainsi, nous allons optimiser une approximation $\hat{G}$ au lieu d'utiliser directement G^*.

Utiliser un modèle de substitution revient à définir une structure de modèle et une stratégie de mise à jour pour améliorer l'approximation au cours de l'optimisation. Dans cette étude, nous avons choisi une structure simple d'interpolation par Inverse Distance Weighting (IDW) [Shepard 1968]. Il faut également décider sur quel espace construire le modèle et comme les contrôleurs ne sont tous évalués que sur l'ensemble d'apprentissage, le choix de l'espace d'entrée est limité. Le modèle peut être par exemple construit sur le génotype des solutions ou sur une description de leurs comportements observés sur Ω_{train}. En nous basant sur divers résultats concluant à l'intérêt d'utiliser des descriptions comportementales en robotique évolutionniste (cf. partie 2.3.2), nous avons choisi de construire le modèle de G^* sur le comportement global des solutions sur les contextes d'apprentissage de Ω_{train}. Afin de justifier ce choix, des investigations supplémentaires sont disponibles dans la partie 5.6 pour comparer l'utilisation de l'espace comportemental et l'espace génotypique dans la construction du modèle de substitution.

Si on appelle $b(c)$ le comportement global correspondant au contrôleur c sur les contextes de Ω_{train}, la capacité de généralisation du contrôleur c est approximée par la valeur $\hat{G}(b(c))$. En supposant qu'une distance comportementale entre individus b_{dist} est définie et que les capacités de généralisation de quelques contrôleurs ont déjà été évaluées sur l'ensemble de validation (au moins 1), il est possible d'interpoler par IDW un modèle de substitution de la capacité de généralisation à partir de ces valeurs.

Soit $\mathcal{C}$ l'ensemble de tous les contrôleurs possibles, soit $\mathcal{C}_{valid}$ l'ensemble des contrôleurs ayant déjà été évalués sur l'ensemble de validation Ω_{valid} et $G^*(c_i)$ la capacité de généralisation exacte correspondant à chaque $c_i \in \mathcal{C}_{valid}$. La valeur prédite par le modèle de substitution $\hat{G}(b(c))$ de la capacité de généralisation pour un contrôleur quelconque c est :

$$\forall c \in \mathcal{C}, \hat{G}(b(c)) = \frac{\sum_{c_i \in \mathcal{C}_{valid}} G^*(c_i)\ b_{dist}(c_i, c)^{-2}}{\sum_{c_i \in \mathcal{C}_{valid}} b_{dist}(c_i, c)^{-2}}.$$

La stratégie de mise à jour du modèle vise à sélectionner les prochains contrôleurs à évaluer sur Ω_{valid} pour améliorer le modèle $\hat{G}$. Nous définissons une stratégie simple qui, à chaque génération, sélectionne l'individu, parmi ceux ayant les plus hautes valeurs de fitness sur l'ensemble d'apprentissage, dont la distance aux contrôleurs

déjà évalués de $\mathcal{C}_{valid}$ est maximale[2]. Ceci permet de sélectionner des contrôleurs à la fois prometteurs d'après leur performance sur Ω_{train} et différents de ceux déjà évalués sur Ω_{valid}.

5.1.2 Critères d'évaluation

Une fois que le modèle de substitution est défini, chaque contrôleur est évalué par 3 critères à maximiser :

1. la performance F sur l'ensemble d'apprentissage Ω_{train} ;
2. la capacité de généralisation approchée sur l'ensemble de validation Ω_{valid} d'après le modèle de substitution $\hat{G}$;
3. le critère de diversité comportementale.

Ce troisième critère est utilisé afin de maintenir une diversité comportementale au sein de la population, ce qui se montre habituellement bénéfique pour assurer une bonne exploration de l'espace de recherche [Trujillo *et al.* 2008, Gomez 2009, Mouret & Doncieux 2011]. Plus de détails sur l'intérêt d'utiliser cet objectif sont disponibles dans la partie 2.3.3 et l'apport de cet objectif sera étudié dans la partie 5.6. Afin de quantifier la diversité d'un comportement par rapport à ceux correspondant aux contrôleurs qui ont déjà été transférés en réalité, nous définissons le critère de diversité comportementale comme suit. Soit $\mathcal{C}_{valid}$ l'ensemble des contrôleurs déjà évalués sur l'ensemble de validation Ω_{valid} et b_{dist} la distance comportementale définie entre individus, la valeur de diversité comportementale $div(c)$ assignée à un contrôleur c quelconque est :

$$div(c) = \min_{c_i \in \mathcal{C}_{valid}} b_{dist}(c,\ c_i)$$

Par l'utilisation d'une distance comportementale, cette valeur de diversité ne dépend pas directement du génotype ou du phénotype des contrôleurs optimisés et favorise les solutions dont les comportements sont les plus différents de ceux des contrôleurs déjà évalués sur Ω_{valid}.

5.1.3 Etapes de l'algorithme

Afin d'initialiser le modèle de substitution $\hat{G}$ de la capacité de généralisation, nous supposons qu'un contrôleur c_0 a déjà été évalué sur l'ensemble de validation avant le début de l'optimisation. A chaque génération, l'algorithme suivant est utilisé comme représenté sur la figure 5.1.

A. Evaluation du contrôleur c :

   A1. calcul de son comportement $b(c)$ et de sa performance $F(c)$ sur l'ensemble d'apprentissage Ω_{train} ;

2. Les deux applications considérées dans ce chapitre utilisent des valeurs de fitness discrètes. On sélectionnera donc l'individu à transférer parmi ceux ayant *la* plus haute valeur de fitness.

A2. évaluation des 2 autres critères par rapport aux contrôleurs déjà évalués de $\mathcal{C}_{valid}$ sur l'ensemble de validation Ω_{valid} (capacité de généralisation approchée $\hat{G}(b(c))$ sur Ω_{valid} et valeur de diversité comportementale) ;

B. Un contrôleur sélectionné par la stratégie de mise à jour est évalué sur l'ensemble de validation Ω_{valid} et le modèle de substitution est modifié en fonction de sa valeur exacte de capacité de généralisation.

C. Application des opérateurs évolutionnistes et génération de la nouvelle population.

Une fois le nombre de générations prévu écoulé, le meilleur contrôleur trouvé par l'optimisation est celui dont la valeur de capacité de généralisation est la plus élevée.

5.1.4 Comparaison avec l'approche par transférabilité

L'approche *ProGAb* étant initialement inspirée de l'approche par transférabilité, les deux méthodes présentent un format similaire : chacune optimise un critère de performance sur un cas simplifié (simulation ou ensemble d'apprentissage) et un autre critère approximé sur le cas-cible (réalité ou ensemble de validation). Il est néanmoins important de noter les différences entre elles. L'approche par transférabilité pose une contrainte sur les solutions par le biais du critère de transférabilité afin de privilégier celles qui sont transférables, quitte à perdre sur le critère de performance : on cherche explicitement un compromis entre performance et transférabilité. Par contre, le seul but de l'approche *ProGAb* est de maximiser la capacité de généralisation sur l'ensemble de validation. La performance sur l'ensemble d'apprentissage ne permet que de guider les solutions dans un premier temps et n'intervient pas pour déterminer quelle solution est optimale.

5.1.5 Validation sur deux expériences robotiques

L'approche *ProGAb* a tout d'abord été testée sur un dispositif expérimental simple de navigation dans un labyrinthe en T pour un robot mobile simulé, inspiré d'une expérience de neurosciences sur la mémoire de travail [Maes *et al.* 2001]. L'autre dispositif expérimental, également étudié en simulation, est une tâche plus complexe de ramassage de balles dans une arène [Doncieux & Mouret 2010, Mouret & Doncieux 2011], où le robot doit effectuer une série de sous-tâches (naviguer, trouver une balle, ramasser la balle, atteindre la corbeille, relâcher la balle, ...) pour obtenir une fitness non nulle. Nous allons maintenant présenter les différentes approches de référence auxquelles nous nous proposons de comparer l'approche *ProGAb*, avant de détailler pour chacune de ces deux applications son dispositif expérimental et les résultats obtenus.

Dans chacune des applications étudiées, nous n'utilisons qu'*un unique contexte d'apprentissage* ($|\Omega_{train}| = 1$) pour l'approche *ProGAb*. Ceci signifie notamment que le modèle de substitution de la capacité de généralisation ne prend en entrée que les

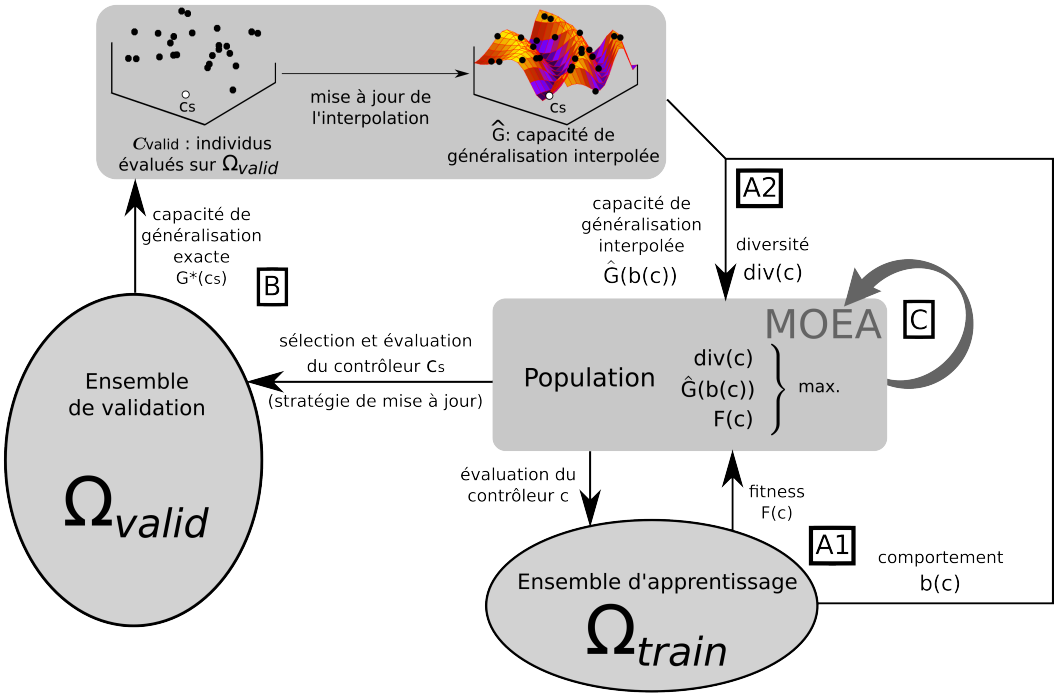

FIGURE 5.1 – Schéma d'une génération de l'approche *ProGAb* – A1. Pour chaque contrôleur c de la population, son comportement $b(c)$ et sa fitness $F(c)$ sont évalués sur l'ensemble d'apprentissage Ω_{train}. A2. Le comportement d'un contrôleur donné permet de calculer la valeur de sa capacité de généralisation prédite par le modèle de substitution $\hat{G}$, ainsi que sa valeur de diversité. B. Suivant la stratégie de mise à jour du modèle, un individu c_s de la population est sélectionné puis évalué sur l'ensemble de validation Ω_{valid}. Ce contrôleur est ajouté à l'ensemble $\mathcal{C}_{valid}$ des contrôleurs déjà testés et sa valeur exacte $G^*(c_s)$ de capacité de généralisation sur Ω_{valid} permet de mettre à jour le modèle de substitution $\hat{G}$ par interpolation. C. Les opérateurs évolutionnistes sont ensuite appliqués aux contrôleurs de la population et un processus de sélection permet de construire la population suivante.

comportements des solutions sur ce contexte d'apprentissage spécifique. Ce point sera discuté dans la partie 5.7.1 après la présentation des différents résultats.

5.2 Approches de référence et détails d'implémentation

5.2.1 Approches de référence

La méthode *ProGAb* est comparée à 4 approches de référence sur deux applications robotiques en simulation, détaillées ici de la plus simple à la plus sophistiquée. Toutes les approches partagent les même ensembles de validation et de test. Les approches de référence diffèrent entre elles par l'ensemble d'apprentissage utilisé.

L'approche *Train_ only* est basée sur un ensemble d'apprentissage ne contenant qu'un unique contexte $|\Omega_{train}| = 1$. Le contexte d'apprentissage choisi est le même que celui utilisé pour la méthode *ProGAb*. Les résultats de cette approche permettront notamment de déterminer si optimiser sur ce contexte précis permet de généraliser à d'autres contextes.

L'approche *Eval_ all* optimise directement la fitness de chaque contrôleur sur l'ensemble de tous les contextes d'évaluation $\Omega_{valid} \subset \Omega_{train}$. En contrepartie du nombre important d'évaluations faites par individu, l'optimisation comptera moins de générations que pour les autres approches.

Les deux approches suivantes sont inspirées de l'article [Panait 2003] : *Fixed-Random-Initial* (*FRI*) et *Random-Per-Generation* (*RPG*). Pour l'approche *FRI*, l'ensemble d'apprentissage contient n contextes ($|\Omega_{train}| > 1$) qui sont choisis aléatoirement dans l'ensemble de validation Ω_{valid} au début de l'optimisation. L'ensemble d'apprentissage reste ensuite fixe.

Pour la dernière approche appelée *RPG*, l'ensemble d'apprentissage contient également n contextes et il est généré aléatoirement à partir de Ω_{valid} à chaque génération. Cette approche est similaire à la méthodologie de Jakobi [Jakobi 1998a] à ceci près que les contrôleurs sont toujours évalués sur un même ensemble d'apprentissage pendant une génération donnée.

Pour chacune de ces approches, les contrôleurs sont évalués par deux objectifs à maximiser : leur fitness moyenne sur les contextes d'apprentissage Ω_{train} et un objectif de diversité. Pour ces différentes expériences de contrôle, l'utilisation d'un critère de diversité suppose qu'une distance comportementale a été définie pour comparer les individus. La valeur de diversité est alors simplement calculée comme la moyenne des distances entre l'individu considéré et le reste de la population. Ainsi, soit $\mathcal{P}$ la population de contrôleurs de taille $|\mathcal{P}|$, soit c le contrôleur évalué et soit b_{dist} la distance comportementale définie entre individus, la valeur de diversité $div(c)$ associée à c s'obtient comme suit :

$$div(c) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} b_{dist}(c,\ p)$$

Enfin, bien que ces différentes approches mettent en jeu des ensembles d'apprentissage de tailles différentes, nous avons fait en sorte de conserver un budget

d'évaluations constant en jouant sur le nombre de générations de l'optimisation. Pour les 4 approches de référence, tous les contrôleurs de la population P étant évalués sur l'ensemble Ω_{train} pendant n_g, le nombre total d'évaluations E_{ctrl} pendant l'optimisation est simplement :

$$E_{ctrl} = |\mathcal{P}| * n_g * |\Omega_{train}| \tag{5.1}$$

Dans le cas de l'approche *ProGAb*, un contrôleur de la population est également évalué sur l'ensemble de validation Ω_{valid} à chaque génération. Dans ce cas, le nombre total d'évaluations E_{ProGAb} se calcule comme suit :

$$E_{ProGAb} = |\mathcal{P}| * n_g * |\Omega_{train}| + n_g * |\Omega_{valid}| \tag{5.2}$$

Les valeurs de paramètres utilisées, ainsi que le budget d'évaluations choisi pour chacune des tâches, sont répertoriées dans les tables 5.2 et 5.6.

5.2.2 Contrôleurs optimisés

Le contrôle du robot dans les deux applications est assuré par un réseau de neurones. A la fois la structure et les paramètres des réseaux sont optimisés en se basant sur un encodage direct simple inspiré de NEAT (Neuro-Evolution of Augmenting Topologies [Stanley & Miikkulainen 2002]) et utilisé dans plusieurs travaux [Doncieux & Mouret 2010, Pinville & Doncieux 2010, Mouret & Doncieux 2011]. Chaque neurone dispose d'un biais dans l'intervalle $[-5, 5]$ et renvoie une valeur dans l'intervalle $[-1, 1]$ en appliquant une fonction de transfert sigmoïde à la somme des valeurs d'entrée.

L'encodage direct utilisé décrit chaque réseau comme un graphe orienté sur lequel 5 opérateurs de mutation sont définis (aucun opérateur de croisement n'est défini) :

- ajout d'une connexion ;
- retrait d'une connexion ;
- ajout d'un neurone ;
- retrait d'un neurone ;
- modification des paramètres (poids ou biais) d'une connexion par mutation polynômiale [Deb 2001b].

5.2.3 Distance comportementale

Dans les deux applications, le comportement du robot est décrit par la séquence temporelle binarisée de ses valeurs de capteurs et d'effecteurs. Pour l'obtenir, les valeurs de chaque capteur (dans l'intervalle $[0, 1]$) sont transformées en 0 quand elles sont inférieures à 0.5 et en 1 sinon. Les effecteurs pouvant prendre des valeurs négatives, le seuillage se fait autour de 0. La distance comportementale entre individus b_{dist} est alors définie comme la distance de Hamming entre deux comportements. Cette distance générique renvoie le nombre de bits qui diffèrent entre les séquences temporelles. Des résultats intéressants ont été obtenus avec cette description comportementale sur la tâche de collecte de balles, où l'utilisation de cette distance

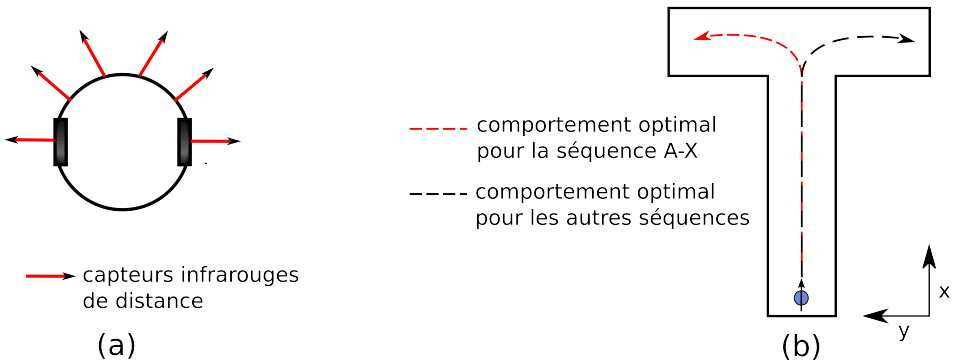

FIGURE 5.2 – (a) Robot mobile utilisé pour la tâche de navigation dans un labyrinthe en T simulé. En plus des capteurs indiqués sur le schéma, le robot dispose d'un capteur pour chacun des 4 stimuli A, B, X, Y. (b) Carte du labyrinthe en T et indication du comportement optimal recherché par séquence de stimuli.

comportementale comme critère de diversité permet d'obtenir les contrôleurs les plus efficaces [Doncieux & Mouret 2010, Mouret & Doncieux 2011]. Pour toutes les approches, le calcul de distance comportementale se fait sur le comportement global observé sur l'ensemble d'apprentissage.

5.3 Application I : navigation dans un labyrinthe

5.3.1 Description

La première application que nous considérons est une extension du problème du "roadsign" [Ziemke & Thieme 2002, Rylatt & Czarnecki 2000] en simulation : un robot navigue dans un labyrinthe en T et doit, suivant un stimulus rencontré pendant la traversée du premier corridor (lumière, son, ...), tourner dans une direction donnée à la jonction. Le dispositif expérimental, ainsi que le type de robot utilisé sont indiqués sur la figure 5.2.

Cette tâche a été modifiée afin de rendre le comportement recherché plus cognitif. Le stimulus que reçoit le robot est une combinaison de 4 stimuli, notés A, B, X et Y. Ce choix est inspiré du test cognitif AX-CPT [Braver *et al.* 1995, Pinville & Doncieux 2010] utilisé pour mettre en évidence des phénomènes de mémoire de travail. Le robot reçoit d'abord un premier stimulus (A ou B), puis un second stimulus (X ou Y) après un délai fixe. Le robot doit alors tourner à gauche à la jonction dans le cas de la séquence A-X, et à droite pour toute autre séquence (A-Y, B-X, B-Y). Cette implémentation de la tâche AX-CPT dans un labyrinthe en T est inspiré d'une expérience venant des neurosciences pour l'étude de la mémoire de travail chez le rat [Maes *et al.* 2001].

L'agent considéré est un robot à 2 deux roues et 10 capteurs : 6 capteurs de distance et 1 capteur par stimulus A, B, X, Y. Un capteur de stimulus renvoie la valeur 1 quand le stimulus correspondant est présenté au robot et 0 sinon. Pour

TABLE 5.1 – Paramètres des ensembles de contextes pour l'application I. Ω_{train} correspond au contexte d'apprentissage utilisé dans les approches *ProGAb* et *Train_ only*. La position initiale sans décalage correspond à un point situé au milieu du premier corridor du labyrinthe (coordonnées [50, 300]). Se reporter à la figure 5.2 pour l'orientation.

Paramètres	Ω_{train}	$\lvert\Omega_{valid}\rvert = 180$	$\lvert\Omega_{test}\rvert = 180$
taille du lab. (mm)	600	[500, 550, 600, 650]	[530, 580, 630, 640]
orientation init. (°)	0	[-30, -15, 0, 15, 30]	[-23, -18, 11, 18, 23]
décalage en x (mm)	0	[-20, 0, 20]	[-10, 10, 30]
décalage en y (mm)	0	[-40, 0, 40]	[-20, 0, 20]

calculer sa valeur globale de fitness, le robot est testé sur les 4 séquences possibles : il obtient 1 point de fitness s'il tourne correctement pour les séquences A-Y, B-X et B-Y, et 3 points pour la séquence A-X. La fitness maximale est donc de 6. La totalité des expériences s'effectue dans un simulateur 2D sans friction ni glissement.

Afin de suivre le protocole expérimental original [Maes *et al.* 2001], les deux moteurs sont désactivés lors de la présentation des stimuli. La tâche complète dure 300 pas de temps et suit le processus suivant, t étant le nombre de pas de temps écoulés :

- $0 < t < 40$: présentation du premier stimulus (A ou B) ;
- $40 < t < 60$: délai, pas de stimulus ;
- $60 < t < 100$: présentation du second stimulus (X ou Y) ;
- $100 < t < 300$: le robot peut bouger et effectue la tâche de navigation dans le labyrinthe.

Afin de définir les ensembles d'apprentissage, de validation et de test, nous définissons 3 paramètres variables par contexte : 1) la taille du labyrinthe ; 2) l'orientation initiale du robot ; 3) les coordonnées initiales du robot (abscisse et ordonnée). Les différentes valeurs utilisées pour définir les ensembles Ω_{train}, Ω_{valid} et Ω_{test} sont répertoriées dans la table 5.1.

5.3.2 Résultats obtenus

Les 5 approches utilisent l'algorithme évolutionniste multi-objectif élitiste NSGA-II [Deb *et al.* 2002b], qui repose sur une sélection par tournoi et une méthode de classement des individus basée sur la relation de dominance (cf. partie 2.2.6.2 pour plus de détails). Ce travail a été implémenté dans le framework Sferes_{v2} [Mouret & Doncieux 2010]. Le code source est disponible sur `http://www.isir.fr/evorob_db`.

Chacune des approches est répétée 20 fois avec un budget total d'évaluations d'environ 3.8 millions d'évaluations. Les budgets exacts, ainsi que les différentes valeur de paramètres utilisées sont indiqués dans la table 5.2. A noter que pour

TABLE 5.2 – Paramètres utilisés dans l'application I ($|\Omega_{valid}| = 180$) afin d'assurer un nombre total d'évaluations d'environ $3.8 \cdot 10^6$ par optimisation (cf. équations (5.1) et (5.2)).

Approches	$\lvert\mathcal{P}\rvert$	n_g	$\lvert\Omega_{train}\rvert$	nombre total d'évaluations
Train_ only	200	19000	1	$3.80 \cdot 10^6$
Eval_ all	”	110	180	$3.96 \cdot 10^6$
FRI	”	1900	10	$3.80 \cdot 10^6$
RPG	”	1900	10	$3.80 \cdot 10^6$
ProGAb	”	10000	1	$3.80 \cdot 10^6$

TABLE 5.3 – Paramètres liés à l'encodage des réseaux de neurones pour l'application I.

Paramètres	Valeurs
nb. min./max. de neurones	3 / 20
nb. min/max. de connexions	5 / 50
prob. d'ajouter un neurone	0.05
prob. d'enlever un neurone	0.05
prob. d'ajouter une connexion	0.05
prob. d'enlever une connexion	0.05
prob. de modifier un poids/biais	0.05

les approches *FRI* et *RPG*, l'ensemble d'évaluation compte 10 contextes. Les paramètres liés à l'encodage des réseaux de neurones utilisés pour le contrôle sont répertoriés dans la table 5.3.

L'ensemble des résultats obtenus avec les 5 approches est indiqué sur le graphe 5.3. Pour chaque méthode sont indiquées les performances des meilleurs individus trouvés (20 répétitions) sur les ensembles d'apprentissage, de validation et de test. La figure 5.4 indique l'évolution de la capacité de généralisation de ces contrôleurs sur l'ensemble Ω_{valid} au cours de l'optimisation. Toutes les comparaisons entre les approches sont effectuées à l'aide de tests de somme de rangs de Wilcoxon (ou test de Mann-Whitney).

Dans cette tâche, il est relativement simple de maximiser la fitness sur un seul contexte. Si on s'intéresse en particulier aux approches *Train_ only* et *ProGAb*, la valeur maximale de fitness est effectivement rapidement atteinte sur l'unique contexte d'apprentissage, au bout de respectivement 55000 et 71000 évaluations en médiane. L'approche *Train_ only* n'obtient pas de contrôleurs capables de généraliser sur l'ensemble de validation avec un nombre médian de contextes réussis sur Ω_{valid} de 3 sur 180. Ce résultat montre que même sur une tâche simple à résoudre dans un

contexte particulier, atteindre la performance maximale sur ce contexte spécifique ne donne pas de garantie de performance sur d'autres contextes.

Sur cette application de navigation, il n'était pas nécessaire d'effectuer un si grand nombre d'évaluations : pour l'ensemble des approches, la capacité de généralisation des contrôleurs trouvés se stabilise au bout de 2 millions d'évaluations sans augmentation ultérieure significative (p-values > 0.35). Malgré cela, l'approche *Eval_ all* trouve des contrôleurs incapables de résoudre au moins un contexte correctement dans 18 optimisations sur 20. Pour cette application, il faudrait un budget d'évaluations plus important pour pouvoir maximiser directement la capacité de généralisation.

Comparé à l'approche simpliste *Train_ only* où un seul contexte d'évaluation est utilisé, les approches *FRI* et *RPG* évaluant chaque contrôleur sur 10 contextes différents se comportent significativement mieux (p-values $< 10^{-3}$), avec des nombres médians respectifs de contextes réussis de 80 et 144 sur 180. Par contre, générer l'ensemble d'évaluation aléatoirement à chaque génération comme le préconise l'approche *RPG* n'a pas d'impact déterminant par rapport à un ensemble de validation fixe utilisé avec *FRI* (p-value $= 0.148$ au bout de 3.8 millions d'évaluations).

L'approche *ProGAb* trouve les contrôleurs aux meilleures capacités de généralisation, avec un nombre médian de contextes réussis de 170 sur les 180 de l'ensemble de validation, ce qui est significativement meilleur que les 4 approches de référence (p-values $< 4 \cdot 10^{-2}$). D'autre part, si on se réfère à la figure 5.4, la capacité de généralisation sur Ω_{valid} augmente plus rapidement avec l'approche *ProGAb*. Pour étudier ce point plus en détail, nous avons sélectionné pour chaque approche les optimisations où le meilleur contrôleur résout plus de 150 contextes de validation (cf. table 5.4). Il s'avère que la méthode *ProGAb* trouve de tels individus deux à trois fois plus rapidement que la méthode *RPG*, avec des nombres d'évaluations médians respectifs de $0.42 \cdot 10^6$ et de $1.21 \cdot 10^6$. Les autres approches n'obtiennent d'aussi bonnes capacités de généralisation que dans moins de 30% des optimisations.

Un dernier point important à mettre en avant concerne la performance sur l'ensemble de test Ω_{test}. En effet, comme nous l'avons indiqué précédemment, même si maximiser la capacité de généralisation sur l'ensemble de validation est déterminant, les contextes de Ω_{valid} font partie des contextes utilisés pour l'évaluation (sauf pour l'approche *Train_ only*). Il est nécessaire, pour prouver que les contrôleurs obtenus ont bien des capacités de généralisation intéressantes, de tester les contrôleurs obtenus sur un ensemble de test disjoint de Ω_{valid}. Il s'avère que l'approche *ProGAb* obtient également les meilleurs résultats sur Ω_{test}, avec un nombre médian de contextes réussis de 150 sur 180, c'est-à-dire sans dégradation significative par rapport aux performances sur l'ensemble de validation (p-value $= 0.45$).

L'approche *ProGAb* permet donc d'améliorer significativement les capacités de généralisation des contrôleurs trouvés sur cette application à la navigation d'un robot mobile simulé dans un labyrinthe en T, et plus rapidement que des approches classiques.

TABLE 5.4 – Nombre d'optimisations N où le meilleur contrôleur de la population résout plus de 150 des 180 contextes de l'ensemble de validation Ω_{valid}. Le nombre d'évaluations nécessaire pour obtenir un tel individu est indiqué : les valeurs de médianes, moyennes et écart-types sont calculés sur les N optimisations concernées. L'approche utilisée a un impact significatif sur le nombre N obtenu d'après un test exact de Fisher (p-value $= 2.63 \cdot 10^{-8}$).

Approche	Nb. d'optimisations (maximum 20)	Nombre d'évaluations (10^6)		
		médiane	moyenne	écart-type
Train_only	0	·	·	·
Eval_all	1	2.59	2.59	0
FRI	6	1.75	1.86	0.95
RPG	12	1.21	1.64	1.12
ProGAb	14	0.42	0.88	0.93

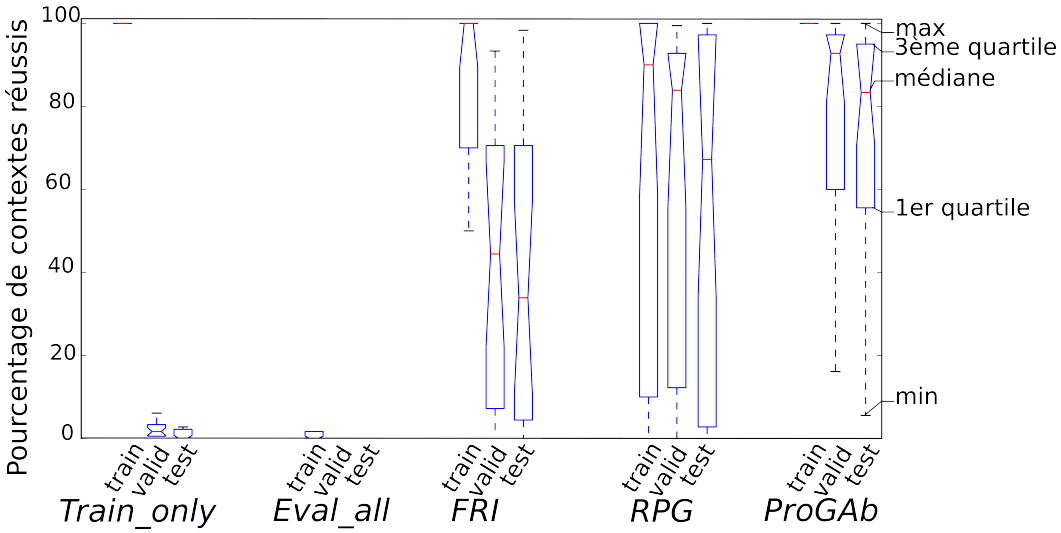

FIGURE 5.3 – Pourcentage de contextes des ensembles Ω_{train}, Ω_{valid} et Ω_{test} réussis par les meilleurs contrôleurs obtenus avec chaque approche pour la tâche de navigation dans un labyrinthe en T (20 répétitions par approche). La méthode *ProGAb* obtient les meilleures capacités de généralisation d'après un test de Welch (p-values < 0.039, cf. annexe 8).

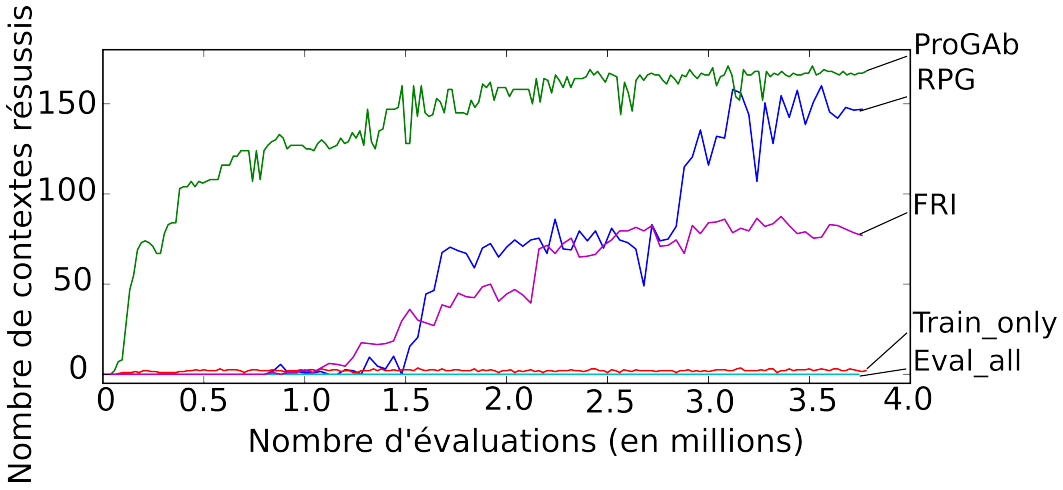

FIGURE 5.4 – Evolution de la capacité de généralisation sur l'ensemble Ω_{valid} des meilleurs contrôleurs obtenus avec chaque approche (valeur médiane sur les 20 répétitions) pour la tâche de navigation dans un labyrinthe en T. Pour obtenir ce graphe, les différents contrôleurs sont évalués sur Ω_{valid} toutes les 10^4 évaluations (hors optimisation). L'ordonnée représente le nombre de contextes réussis (i.e. où le contrôleur a obtenu la fitness maximale de 6) : 180 contextes au maximum.

5.4 Application II : collecte de balles dans une arène

5.4.1 Description

La tâche de collecte de balles implémentée ici (cf. figure 5.5) est principalement basée sur le dispositif expérimental décrit dans l'article [Doncieux & Mouret 2010, Mouret & Doncieux 2011]. Un robot doit naviguer dans une arène, trouver 4 balles, et les amener une par une dans une corbeille. Pour résoudre cette tâche, le robot doit pouvoir naviguer dans l'arène et modifier son comportement quand il détecte une balle. Plus précisément, on peut distinguer plusieurs sous-tâches à résoudre : naviguer pour trouver une balle, récupérer cette balle, naviguer pour trouver la corbeille, poser la balle dans la corbeille, naviguer pour trouver une deuxième balle, ... La fonction de fitness utilisée ne récompense pas le contrôleur évalué pour ces différentes étapes : la valeur de la fitness correspond au nombre de balles dans la corbeille à la fin de la simulation. Une telle fonction de fitness rend la tâche d'autant plus complexe à résoudre par un processus d'optimisation.

L'agent considéré est un robot à deux roues possédant 10 capteurs : 3 capteurs de distance situés à l'avant, 2 capteurs de contact binaires, 2 capteurs par tranche pour détecter les balles[3], 2 capteurs par tranche pour détecter la corbeille[3] et un capteur binaire activé seulement si le robot est en train de porter une balle. Le robot

3. L'un des capteurs par tranche détecte les éléments situés à gauche du robot, l'autre détecte les éléments situés à droite. Cf. figure 5.5 pour une illustration du champ de détection.

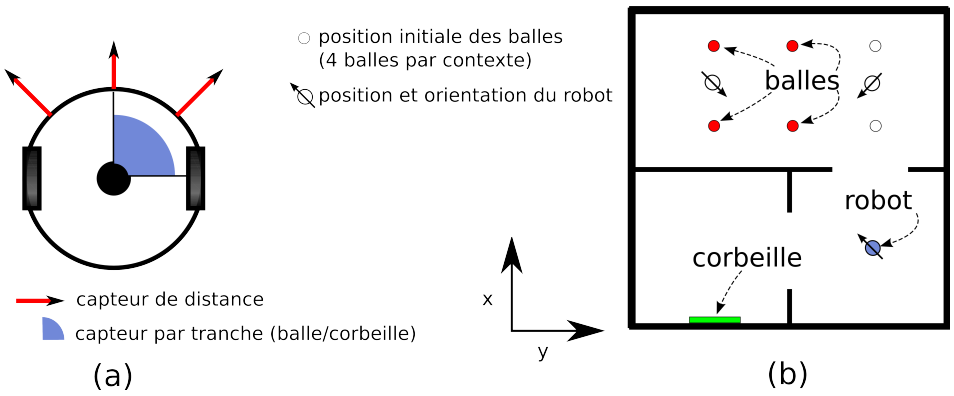

FIGURE 5.5 – (a) Présentation de la tâche de collecte de balles. En plus de ses 3 capteurs de distance, le robot est équipé de 2 capteurs détectant les balles et de 2 capteurs détectant la corbeille. Le champ de détection des capteurs droits est indiqué sur la figure. Celui des capteurs gauches est symétrique. Un réseau de neurones contrôle les vitesses des deux roues motorisées, ainsi qu'un moteur de ramassage de balles. (b) Arène où le robot évolue. Les 6 positions possibles des balles et quelques positions et orientations initiales du robot sont schématisées.

possède également 3 effecteurs : les deux roues et un moteur de récupération de balle. La valeur associée à ce dernier moteur doit être supérieure à 0.5 pour ramasser une balle et la conserver, et inférieure à 0.5 pour la relâcher. Si le robot détient déjà une balle, il ne peut pas en ramasser d'autres.

La fonction de fitness compte le nombre de balles dans la corbeille à la fin des 3000 pas de temps de l'expérience. La totalité des expériences s'effectue dans un simulateur 2D sans friction ni glissement.

Les paramètres définissant un contexte ω pour cette application sont les suivants : 1) la taille de l'arène ; 2) l'orientation initiale du robot ; 3) les coordonnées initiales du robot (abscisse et ordonnée) ; 4) le positionnement initial des 4 balles. Il y a 6 positions de balles possibles comme indiqué sur le schéma 5.5, soit 15 positionnements possibles. L'ensemble des valeurs des différents paramètres est répertorié dans la table 5.5.

5.4.2 Résultats obtenus

Les 5 approches utilisent l'algorithme évolutionniste multi-objectif NSGA-II [Deb *et al.* 2002b], qui repose sur une sélection élitiste par tournoi et une méthode de classement des individus basée sur la relation de dominance (cf. section 2.2.6.2 pour plus de détails). Ce travail a été implémenté dans le framework Sferes$_{v2}$ [Mouret & Doncieux 2009a]. Le code source est disponible sur `http://www.isir.fr/evorob_db`.

Chacune des approches est répétée 20 fois avec un budget total d'environ 1.2 millions d'évaluations. Les budgets exacts, ainsi que les différentes valeur de paramètres utilisées sont indiqués dans la table 5.6. A noter que pour les approches

TABLE 5.5 – Paramètres des ensembles de contextes pour l'application II. Ω_{train} correspond au contexte d'apprentissage utilisé dans les approches *ProGAb* et *Train_ only*. Les coordonnées initiales sont exprimées en pourcentage de la taille de l'arène. Se reporter à la figure 5.5 pour l'orientation.

Paramètres	Ω_{train}	$\lvert\Omega_{valid}\rvert = 30$	$\lvert\Omega_{test}\rvert = 600$
taille de l'arène (mm)	600	600	650
orientation initiale (°)	0	0	[-180, -90, 45, 135]
abscisse initiale	0.75	[0.25, 0.75]	[0.25, 0.75]
ordonnée initiale	0.75	0.75	[0.25, 0.40, 0.50, 0.60, 0.75]
nb. de config. des balles	1	15	15

TABLE 5.6 – Paramètres utilisés dans l'application II ($\lvert\Omega_{valid}\rvert = 30$) afin d'assurer un nombre total d'évaluations de $1.2 \cdot 10^6$ par optimisation (cf. équations (5.1) et (5.2)).

Approches	$\lvert\mathcal{P}\rvert$	n_g	$\lvert\Omega_{train}\rvert$	nombre total d'évaluations
Train_ only	200	6000	1	$1.20 \cdot 10^6$
Eval_ all	”	200	30	$1.20 \cdot 10^6$
FRI	”	1500	4	$1.20 \cdot 10^6$
RPG	”	1500	4	$1.20 \cdot 10^6$
ProGAb	”	5500	1	$1.27 \cdot 10^6$

TABLE 5.7 – Paramètres liés à l'encodage des réseaux de neurones pour l'application II.

Paramètres	Valeurs
nb. min./max. de neurones	10 / 30
nb. min/max. de connexions	50 / 250
prob. d'ajouter un neurone	0.15
prob. d'enlever un neurone	0.05
prob. d'ajouter une connexion	0.05
prob. d'enlever une connexion	0.05
prob. de modifier un poids/biais	0.15

FRI et *RPG*, l'ensemble d'évaluation compte 4 contextes. Les paramètres liés à l'encodage des réseaux de neurones utilisés pour le contrôle sont répertoriés dans la table 5.7.

L'ensemble des résultats obtenus avec les 5 approches est indiqué sur le graphe 5.6. Pour chaque méthode sont indiquées les performances des meilleurs individus trouvés (20 répétitions) sur les ensembles d'apprentissage, de validation et de test. La figure 5.7 indique l'évolution de la capacité de généralisation de ces contrôleurs sur l'ensemble Ω_{valid} au cours de l'optimisation. Toutes les comparaisons entre les approches sont effectuées à l'aide de tests de somme de rangs de Wilcoxon (ou test de Mann-Whitney).

Cette seconde tâche est bien plus complexe à résoudre que la précédente : l'approche *Train_ only* ne trouve pas toujours des contrôleurs optimaux sur le contexte d'apprentissage et les 4 balles sont collectées dans seulement 13 des 20 optimisations. Ce résultat est cohérent avec d'autres travaux effectués sur ce dispositif expérimental [Doncieux & Mouret 2010, Mouret & Doncieux 2011].

Comme dans la première application, les approches *Train_ only* et *Eval_ all* correspondent à des capacités de généralisation faibles, avec respectivement 0.63 et 1.00 balles collectées en moyenne sur l'ensemble des contextes de validation (sur 4 balles). Globalement, les performances des approches *Eval_ all*, *FRI* et *RPG* ne sont pas très significativement différentes (p-values > 0.07), avec respectivement 1.00, 1.70 et 2.03 balles collectées en moyenne sur l'ensemble des contextes de Ω_{valid}.

Seule l'approche *ProGAb* obtient des résultats significativement meilleurs avec des contrôleurs pouvant ramasser en moyenne 2.63 sur les contextes de validation (p-values < 0.018). Il s'avère que la performance sur Ω_{test} est également bonne avec une moyenne de balles collectées de 2.18 par contexte. La différence n'est pas significative (p-value $= 0.107$). En comparaison, les performances de *RPG*, la deuxième meilleure approche sur cette application, sont de 2.07 balles collectées en moyenne sur Ω_{valid} et de seulement 0.65 balles collectées en moyenne sur Ω_{test} (dégradation significative : p-value $= 6.5 \cdot 10^{-3}$).

Pour cette seconde application sur une tâche complexe de collecte de balles dans

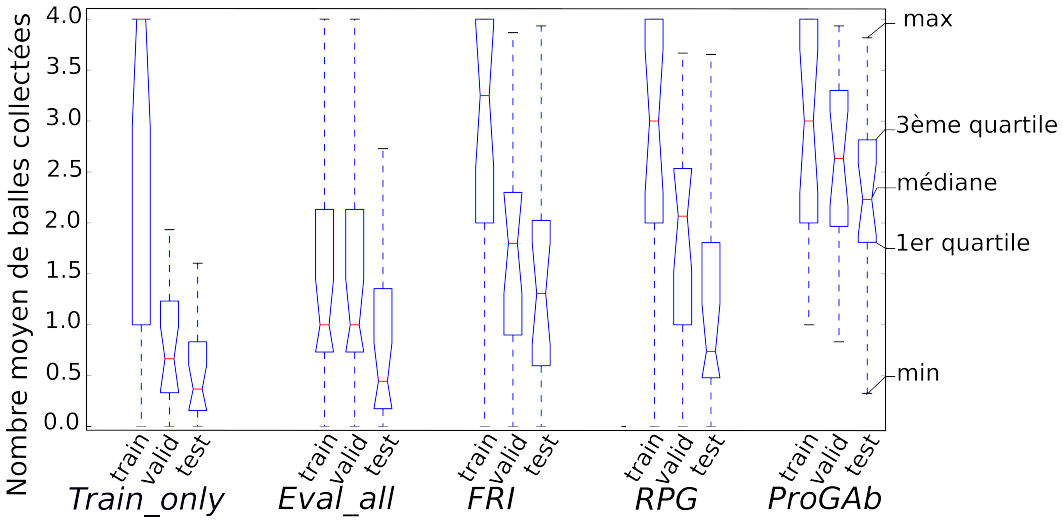

FIGURE 5.6 – Nombre moyen de balles collectées par contexte sur les ensembles Ω_{train}, Ω_{valid}, Ω_{test} par les meilleurs contrôleurs trouvés par chaque méthode sur la tâche de collecte de balle (20 répétitions par approche). La méthode *ProGAb* obtient les meilleures capacités de généralisation d'après un test de Welch (p-values < 0.018, cf. annexe 8).

une arène, l'approche *ProGAb* semble également être une méthode performante pour améliorer les capacités de généralisation des contrôleurs.

5.5 Rappels des résultats principaux

Ces résultats sur deux dispositif expérimental robotiques simulés sont relativement cohérents et permettent de valider 4 conclusions principales :

1. Optimiser les contrôleurs sur un unique contexte conduit rarement à des solutions dotées de capacités de généralisation.
2. Optimiser les contrôleurs sur un très grand nombre de contextes n'est pas non plus efficace, car nécessitant un nombre élevé d'évaluations.
3. Optimiser les contrôleurs sur un nombre limité de contextes (10 pour la première application, 4 pour la seconde) permet d'obtenir des contrôleurs aux capacités de généralisation moyennes sans imposer un nombre trop important d'évaluations par génération.
4. L'approche *ProGAb* obtient des contrôleurs aux capacités de généralisation élevées tout en assurant un nombre total d'évaluations faible via l'usage d'un modèle de substitution. Ceci valide notamment l'idée de construire le modèle de substitution de la capacité de généralisation à partir du comportement des solutions sur un unique contexte d'apprentissage.

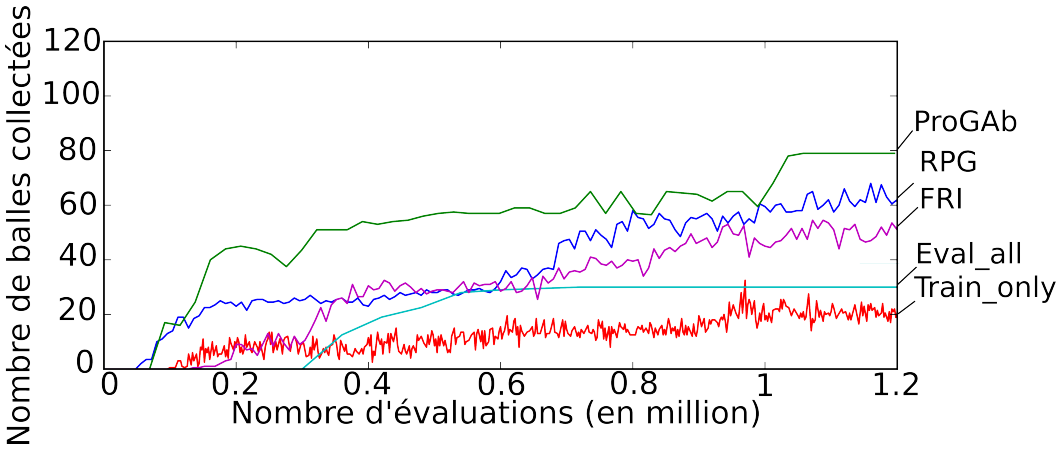

FIGURE 5.7 – Evolution de la capacité de généralisation sur l'ensemble Ω_{valid} des meilleurs contrôleurs obtenus avec chaque approche (valeur médiane sur les 20 répétitions). Pour obtenir ce graphe, les différents contrôleurs sont évalués sur Ω_{valid} toutes les 10^4 évaluations (hors optimisation). L'ordonnée représente le nombre total de balles collectées sur tous les contextes de Ω_{valid} : 30 contextes comptant chacun 4 balles, d'où 120 balles collectées au maximum.

5.6 Investigations supplémentaires

Maintenant que l'intérêt de l'approche *ProGAb* a été mis en avant sur deux applications par rapport à des méthodes plus classiques, nous allons tester 4 points :

Q1. la sélection assurée par la stratégie de mise à jour utilisée apporte-t-elle un gain de performances ?

Q2. le critère de diversité est-il nécessaire ?

Q3. est-il obligatoire de maximiser la performance sur le contexte d'apprentissage de Ω_{train} ou suffit-il de maximiser la capacité de généralisation approchée sur l'ensemble de validation Ω_{valid} ?

Q4. peut-on construire le modèle de substitution directement à partir du génotype des solutions ?

5.6.1 Stratégie de mise à jour

Pour répondre à la question Q1, deux variantes de l'approche *ProGAb* ont été implémentées et testées sur les deux applications précédentes (cf. table 5.8 pour un récapitulatif) :

- *MaxDivStrat.*, où la stratégie de mise à jour sélectionne systématiquement l'individu de la population ayant la valeur de diversité maximale pour l'évaluer sur Ω_{valid} à chaque génération ;
- *RandomStrat.*, où la stratégie de mise à jour sélectionne aléatoirement dans la population le nouvel individu à évaluer sur Ω_{valid} à chaque génération.

TABLE 5.8 – Récapitulatif des différentes variantes implémentées : critères optimisés et caractéristiques principales.

Approches	Critères			Caractéristiques
	F	G	Div.	
MaxDivStrat.	x	x	x	l'individu à la diversité maximale est testé sur Ω_{valid}
RandomStrat.	x	x	x	un individu choisi aléatoirement dans la population est testé sur Ω_{valid}
NoTrain		x	x	
NoDiv	x	x		
GenotypeBased		x	x	modèle de substitution et diversité basés sur une distance génotypique

TABLE 5.9 – Capacités de généralisation sur Ω_{valid} et Ω_{test} des meilleurs individus obtenus suivant la stratégie de mise à jour du modèle de substitution utilisée : nombre de contextes résolus pour l'application I et nombre moyen de balles ramassées par contexte pour l'application II.

Approches	Application I (/ 180)		Application II (/ 4)	
	Ω_{valid}	Ω_{test}	Ω_{valid}	Ω_{test}
ProGAb	170	150	2.63	2.18
MaxDivStrat.	15	9	0.03	0.13
RandomStrat.	137	130	0.60	0.57

Pour ces deux approches, 10 optimisations ont été effectuées sur chacun des deux applications étudiées précédemment avec les même budgets d'évaluations. Les résultats sont représentés respectivement sur la figure 5.8 pour l'application à la navigation dans un labyrinthe et sur la figure 5.9 pour la tâche de collecte de balles. Les capacités de généralisation sur les ensembles de validation et de test des meilleurs individus obtenus sont récapitulés dans le tableau 5.9.

En comparant les approches *ProGAb* et *RandomStrat.*, la différence de résultats n'est pas très significative sur l'application I (p-values > 0.103), mais l'écart est plus franc sur l'application II (p-values < 0.014). L'approche *ProGAb* obtient néanmoins des contrôleurs avec de meilleures capacités de généralisation en médiane dans les deux cas (cf. table 5.9). Ces résultats indiquent donc qu'il semble relativement efficace de biaiser la construction du modèle de substitution vers les individus performants sur l'ensemble d'apprentissage et différents des solutions déjà évaluées sur Ω_{valid}.

Dans les deux cas, étant donné que l'un des objectifs optimisés est la performance sur Ω_{train}, il est plus facile pour l'approche de trouver des individus performants sur le contexte d'apprentissage et qui généralisent, plutôt que des individus qui généralisent sans être performants sur Ω_{train}. Or, l'approche *ProGAb* cherche à construire un modèle de substitution principalement fiable pour les solutions ef-

ficaces sur Ω_{train}, alors que la stratégie de mise à jour aléatoire *RandomStrat.* ne privilégie aucune zone de l'espace de recherche. Le modèle de substitution interpolé avec la stratégie aléatoire va donc potentiellement couvrir une zone de comportements plus large, mais, pour un même budget d'évaluations, il risque d'être moins précis pour les solutions performantes sur Ω_{train} qu'un modèle utilisant la stratégie de l'approche *ProGAb*. Ainsi, dans le cas de la stratégie *RandomStrat.*, on biaise l'optimisation vers des solutions performantes sur Ω_{train}, alors que le modèle de substitution n'est pas forcément très fiable pour ces solutions, ce qui peut expliquer la différence de résultats observée.

Si cette hypothèse est vraie, chercher à maximiser seulement la dispersion des contrôleurs évalués sur Ω_{valid} ne devrait pas bien fonctionner. L'approche *MaxDivStrat.* obtient en effet des résultats très mauvais par rapport à l'approche initiale *ProGAb* et ce pour les deux applications (p-value $< 1.27 \cdot 10^{-6}$). En évaluant systématiquement l'individu de la population qui maximise la diversité comportementale, on cherche à construire un modèle de substitution sur l'ensemble de l'espace comportementale qui compte plus de 2^{1000} comportements possibles. On se rend donc facilement compte que construire un tel modèle va nécessiter beaucoup plus d'expériences que construire un modèle local fiable uniquement pour les individus bons sur Ω_{train}.

Ces résultats valident le choix de la stratégie de mise à jour du modèle de substitution utilisée pour l'approche *ProGAb*.

5.6.2 Critères d'optimisation

Pour répondre maintenant aux questions Q2 et Q3, deux variantes de l'approche *ProGAb* ont été implémentées et testées sur les deux applications précédentes (cf. table 5.8 pour un récapitulatif) :

- la variante *NoTrain*, n'optimisant pas de critère de performance sur Ω_{train} ; dans ce cas, l'optimisation se base uniquement sur le modèle de substitution et sur le critère de diversité comportementale) ;
- la variante *NoDiv*, sans critère de diversité comportementale

Pour chacune de ces approches, 10 optimisations ont été effectuées sur les deux applications étudiées précédemment avec les même budgets d'évaluations. Les résultats sont respectivement récapitulés sur la figure 5.8 pour l'application à la navigation dans un labyrinthe et sur la figure 5.9 pour la tâche de collecte de balles.

L'approche *ProGAb* est significativement meilleure sur les 2 applications que la variante *NoTrain*, aussi bien sur l'ensemble de validation (p-values $= 0.047$ sur l'application I, $= 2.33 \cdot 10^{-7}$ sur l'application II) que sur l'ensemble de test (p-values < 0.034 sur l'application I, $< 1.76 \cdot 10^{-6}$ sur l'application II). La variante *NoTrain* construit un modèle de substitution sur le même espace comportemental que l'approche *ProGAb*, mais n'optimise que la capacité de généralisation des solutions prédite par ce modèle ainsi que le critère de diversité comportementale.

De tels résultats indiquent qu'il est préférable de maximiser la performance des contrôleurs sur un ensemble d'apprentissage, même si, au terme de l'optimisation,

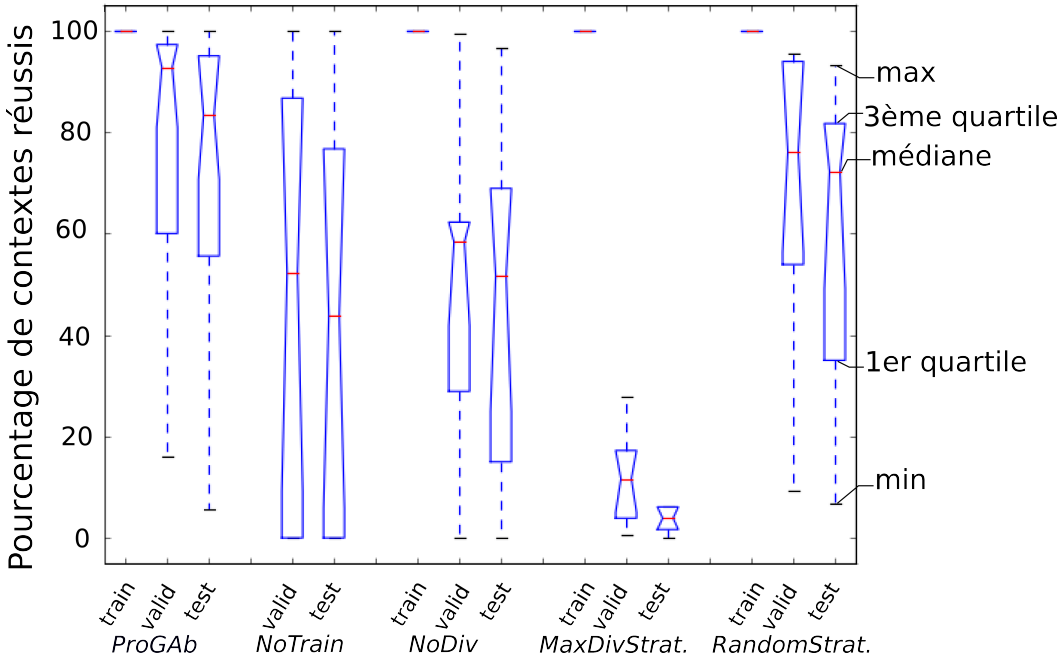

FIGURE 5.8 – Résultats des différentes variantes sur l'application de navigation dans un labyrinthe en T : fitness moyenne sur l'ensemble d'apprentissage et pourcentage de contextes réussis sur les ensembles de validation et de test. La méthode *ProGAb* obtient les meilleures capacités de généralisation d'après un test de Welch (p-values < 0.047).

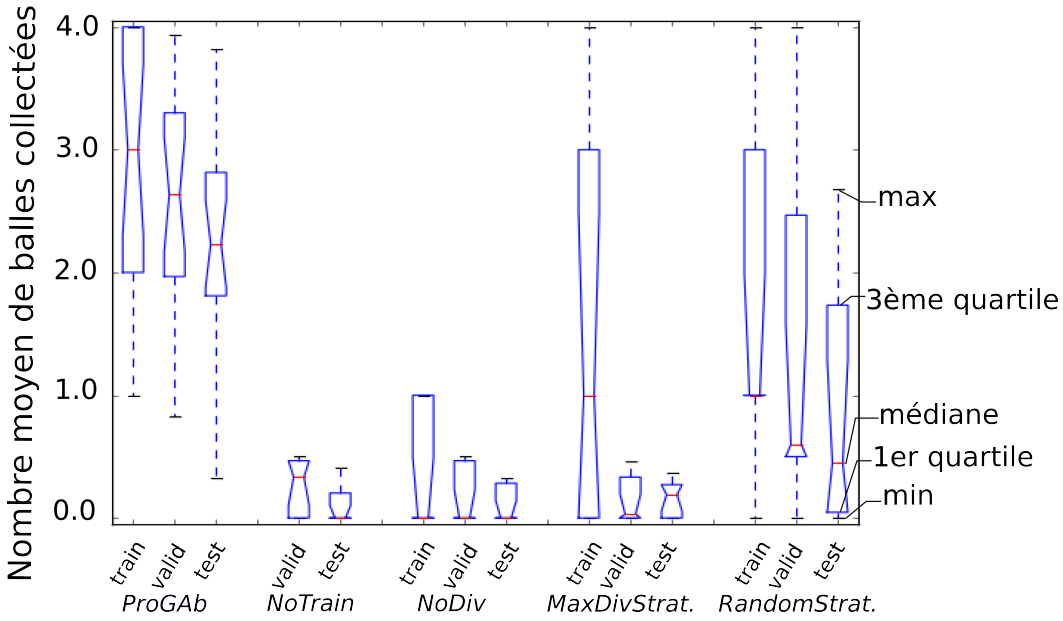

FIGURE 5.9 – Résultats des différentes variantes sur l'application de ramassage de balles : nombre moyen de balles collectées par contexte des ensembles d'apprentissage, de validation et de test. La méthode *ProGAb* obtient les meilleures capacités de généralisation d'après un test de Welch (p-values < 0.034).

seule la capacité de généralisation sur Ω_{valid} est intéressante. En effet, il est probable que des contrôleurs aléatoires aient une performance nulle ou uniforme sur l'ensemble de validation : chercher seulement à maximiser la capacité de généralisation sur Ω_{valid} risque de ne pas fournir de gradient utile pendant une partie importante du processus d'optimisation. En cherchant à résoudre le contexte d'apprentissage, un gradient est fourni à l'optimisation permettant une sélection des solutions sur leur performance d'apprentissage jusqu'à ce qu'un individu ayant des capacités de généralisation non nulles soit découvert.

De même, l'approche *ProGAb* obtient de meilleurs résultats que la variante *NoDiv*, sur les ensembles de validation et de test (p-values < 0.029 pour l'application I, p-values $< 1.93 \cdot 10^{-6}$ pour l'application II). La nécessité d'inclure un critère de diversité comportementale est évidente, surtout dans la tâche de collecte de balles (seulement 3 optimisations trouvent des individus ramassant au moins une balle avec la variante *NoDiv*). Dans le cas contraire, l'optimisation est très sensible aux optima locaux et peut donc gaspiller une grande partie de son budget d'évaluations à échapper aux gradients locaux du paysage de fitness.

Ces différents résultats valident le schéma d'optimisation proposé par l'approche *ProGAb* afin d'obtenir des contrôleurs avec de bonnes capacités de généralisation : (1) performance sur l'ensemble d'apprentissage ; (2) capacité de généralisation approchée sur l'ensemble de validation ; (3) critère de diversité comportementale.

5.6.3 Modèle de substitution basé sur le génotype

L'utilisation classique des modèles de substitution dans un algorithme évolutionniste est d'approximer la fonction de fitness de l'environnement réel à partir du génotype des solutions, sans passer par une simulation intermédiaire. Nous avons choisi lors de la définition de l'approche *ProGAb* de construire le modèle de substitution sur un espace comportemental obtenu dans un environnement simplifié (le contexte d'apprentissage), le lien entre génotype et performance étant *a priori* moins direct qu'entre comportement et performance.

Pour étudier ce point plus en détail, nous avons implémenté la variante *GenotypeBased* où le modèle de substitution est construit directement à partir du génotype des solutions. Cette variante se différencie de l'approche *ProGAb* par deux aspects :

- les solutions ne sont évaluées qu'à partir du modèle de substitution (pas de notion de contexte d'apprentissage) ;
- la distance entre individus b_{dist} compare les génotypes des solutions.

Dans les deux applications considérées, les génotypes sont des graphes orientés développés en réseaux de neurones. Définir une fonction de distance viable entre graphes est un problème d'autant plus difficile que certaines mesures impliquent un coût computationnel important. A l'opposé, utiliser des mesures très simples (par exemple, la différence de nombre de nœuds) peut gêner l'optimisation en introduisant des pressions néfastes directement sur le génotype des solutions (comme l'augmentation du nombre de neurones).

Nous avons utilisé une distance inspirée des travaux sur l'encodage NEAT

(Neuro-Evolution of Augmenting Topologies [Stanley & Miikkulainen 2002]). L'idée principale de NEAT est de partir de réseaux initialement simples (feed-forward complètements connectés) et de les complexifier au fur et à mesure de l'optimisation. Chaque graphe est représenté par un ensemble de gènes, un gène donné correspondant à un nœud ou une connexion. Plusieurs types de mutations sont définis sur le graphe :

- la modification du poids d'une connexion ;
- l'ajout d'une connexion entre deux nœuds initialement non connectés ;
- l'ajout d'un nœud.

Pour ajouter un nœud, une des connexions du graphe est inactivée et les nœuds qu'elle connectait deviennent chacun connecté au nouveau nœud : deux connexions sont donc également ajoutées au graphe.

A chaque mutation modifiant la structure des graphes, un numéro d'innovation global est incrémenté et assigné au nouveau nœud ou à la nouvelle connexion. Le numéro d'innovation d'un gène n'est pas modifié quand il est transmis sans mutation d'un parent à un nouvel individu. Il est alors possible de comparer deux graphes : plus les graphes auront des gènes de même numéros d'innovation, plus ils seront proches. Considérons deux graphes G_1 et G_2. On appelle $\mathcal{D}_1$ (resp. $\mathcal{D}_2$) l'ensemble des gènes de G_1 (resp. G_2) ne correspondant à aucun gène de G_2 (resp. G_1). Soit $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$ l'union de ces deux ensembles et $\mathcal{I}$ l'ensemble des gènes de G_1 ayant une correspondance dans G_2[4]. $\overline{\Delta_w}$ étant la différence moyenne de poids observée entre les gènes de $\mathcal{I}$ dans G_1 et les gènes de $\mathcal{I}$ dans G_2, la distance génotypique b_{dist} entre les deux graphes G_1 et G_2 est calculée comme suit :

$$b_{dist}(G_1,\ G_2) = c_1 \frac{|\mathcal{D}|}{|\mathcal{D}| + |\mathcal{I}|} + c_2 \overline{\Delta_w}$$

Cette distance est élevée si les deux graphes ont beaucoup de gènes disjoints ou si leurs gènes communs ont des valeurs de poids très différentes. Les valeurs des paramètres c_1 et c_2 sont fixées par défaut comme dans l'article [Stanley & Miikkulainen 2002] : $c_1 = 1$ et $c_2 = 0.4$.

Cette distance est utilisée dans la variante *GenotypeBased* pour construire le modèle de substitution par interpolation IDW et également afin de calculer le critère de diversité. Pour rester cohérent avec l'approche *ProGAb*, la diversité est calculée comme suit. Soit $\mathcal{C}_{valid}$ l'ensemble des contrôleurs déjà évalués sur l'ensemble de validation Ω_{valid} et b_{dist} la distance génotypique entre individus, la valeur de diversité $div(c)$ assignée à un contrôleur c quelconque est :

$$div(c) = \min_{c_i \in \mathcal{C}_{valid}} b_{dist}(c,\ c_i)$$

Il est également nécessaire de définir une stratégie de mise à jour pour le modèle de substitution : l'individu évalué sur l'ensemble de validation à chaque génération est simplement celui dont la valeur de diversité est maximale.

4. Chaque nœud de G_1 appartient soit à l'ensemble $\mathcal{D}_1$, soit à l'ensemble $\mathcal{I}$. De même, chaque nœud de G_2 appartient soit à l'ensemble $\mathcal{D}_2$, soit à l'ensemble $\mathcal{I}$.

Cette variante a été testée 10 fois sur chacune des deux applications. Il s'avère que même pour la tâche relativement simple de navigation dans un labyrinthe, l'approche ne trouve au mieux qu'un comportement où le robot tourne toujours à droite ou toujours à gauche (avec une fitness de 3 sur 6 sur tous les contextes). En conséquence, aucun contexte n'est résolu correctement que ce soit dans l'ensemble de validation ou dans l'ensemble de test. De même pour la tâche de ramassage de balles, aucune des optimisations n'a permis de trouver un individu ramenant au moins une balle à la corbeille.

Deux phénomènes peuvent expliquer ces mauvais résultats. Tout d'abord un modèle de substitution n'est pas forcément efficace pour générer un gradient pour l'optimisation : sans *a priori* sur l'espace de recherche, un grand nombre d'expériences peut parfois être nécessaire pour trouver des solutions un tant soit peu efficaces. Pour résoudre ce problème, certains travaux optimisant directement sur un modèle de substitution utilisent une solution non triviale de performance non nulle pour initialiser le processus d'optimisation [Hemker *et al.* 2009]. D'autre part, même si la distance génotypique de NEAT se justifie bien pour comparer des graphes dans un contexte d'optimisation, elle ne donne aucune information fiable sur la dynamique des réseaux de neurones obtenus : elle peut potentiellement distinguer des réseaux aux comportements identiques et trouver très proches des réseaux tout à fait différents [Mouret & Doncieux 2011]. D'une manière générale, en utilisant une distance génotypique il est difficile de comprendre ce qui est réellement optimisé avec possiblement des pressions néfastes appliquées directement sur le génotype.

5.7 Discussion

5.7.1 Prédire la capacité de généralisation à partir d'un contexte

L'approche *ProGAb* propose de construire un modèle de substitution de la capacité de généralisation des solutions sur l'ensemble de validation Ω_{valid} au cours de l'optimisation. Dans les applications présentées, la valeur prédite par le modèle est calculée à l'aide d'une distance comportementale comparant le comportement du contrôleur considéré à ceux des contrôleurs déjà évalués sur *un unique ensemble d'apprentissage* ($|\Omega_{train}| = 1$). Le modèle de substitution cherche donc à approximer la relation entre le comportement d'une solution sur Ω_{train} et sa capacité de généralisation sur Ω_{valid}. Il s'avère que l'approche *ProGAb* donne de bons résultats en terme de généralisation, validant ainsi le principe d'aider l'optimisation par l'intermédiaire d'un modèle de substitution. Dans notre implémentation, l'ensemble d'apprentissage ne compte pourtant qu'un seul contexte ω, ce qui signifie donc qu'il est possible de déduire la capacité de généralisation d'une solution à partir de son comportement sur un unique contexte. Ce résultat semble à première vue contre-intuitif : l'estimation de la capacité de généralisation ne devrait être possible qu'en évaluant la solution sur plusieurs contextes.

Ce fait peut être expliqué en s'intéressant plus en détail à la stratégie de mise à jour employée pour construire le modèle : les individus évalués sur Ω_{valid} sont choisis

parmi les solutions les plus performantes sur Ω_{train}. Le modèle de substitution ne va donc pas faire une approximation globale de la relation entre le comportement sur Ω_{train} et la généralisation sur Ω_{valid}, mais seulement localement pour les individus bons sur Ω_{train}. Grossièrement, le modèle doit ensuite distinguer parmi ces individus ceux qui généralisent de ceux qui ne généralisent pas.

Un individu bon sur Ω_{train} (donc pouvant résoudre au moins en partie la tâche) ne va pas généraliser efficacement si son comportement est trop dépendant des spécifités du contexte d'apprentissage qui varient dans les autres contextes. Il est donc assez vraisemblable qu'une description comportementale puisse distinguer les individus exploitant les spécifités du contexte d'apprentissage. Si cette hypothèse est vraie, le comportement sur un unique contexte d'apprentissage peut permettre de distinguer les individus pouvant généraliser à d'autres contextes de ceux qui ont sur-appris. Ceci explique pourquoi l'approche *ProGAb* peut fonctionner dans les applications étudiées en utilisant un ensemble d'apprentissage de taille 1.

5.7.2 Espace génotypique ou comportemental

Les modèles de substitution sont généralement utilisés pour approximer la fonction de fitness directement à partir du génotype des solutions. Ici, nous avons choisi d'utiliser un espace d'entrée comportemental afin de construire le modèle de la capacité de généralisation, le lien entre généralisation et comportement nous semblant plus direct qu'entre généralisation et génotype. Afin de vérifier cette hypothèse, nous avons tenté une implémentation de l'approche dans la partie précédente en nous basant sur des distances génotypiques : les résultats de cette variante sont très significativement inférieurs à ceux obtenus avec l'approche *ProGAb*.

Une première explication est possible en s'intéressant aux génotypes que nous avons utilisés. Dans les deux applications considérées, nous avons optimisé des graphes développés par la suite en réseaux de neurones. Aucune contrainte sur la topologie ou le nombre de nœuds des graphes n'étant définie, leur comparaison se révèle être un problème compliqué. De nombreuses fonctions de distance peuvent être définies avec des complexités différentes : taille du plus grand sous-graphe commun (problème NP-complet), différence du nombre de nœuds ou de connexions (plus simple), comparaison des connectivités, ... Nous avons choisi ici d'implémenter la variante basée sur l'espace génotypique à l'aide de la distance utilisée dans NEAT (Neuro-Evolution of Augmenting Topologies [Stanley & Miikkulainen 2002]) qui prend en compte les historiques de mutation pour comparer les individus. Néanmoins, il est difficile de justifier une distance par rapport à une autre, de même que de prévoir les pressions possibles induites sur l'optimisation des génotypes. Dans le cas qui nous intéresse, la distance de NEAT se semble pas convenir.

Néanmoins, au-delà de considérations sur la structure du génotype, il est évident qu'une distance génotypique ne prendra, par définition, jamais en compte la dynamique du réseau de neurones dans l'environnement d'évaluation. Or, une distance génotypique peut typiquement prédire que deux graphes sont différents, d'après leur topologie par exemple, alors que les comportements obtenus lors de l'évaluation sont

très proches et vice-versa [Mouret & Doncieux 2011]. Suivant le génotype utilisé, il peut donc être intéressant de comparer les solutions dans un espace comportemental afin de ne distinguer que les individus effectivement différents au regard de la tâche à résoudre.

5.7.3 Généraliser sur des robots physiques

Les deux applications considérées ici, bien que non triviales, ont été étudiées dans un dispositif expérimental complètement simulé. Il est donc intéressant maintenant de discuter de l'utilisabilité de l'approche *ProGAb* sur un robot physique. Quelles sont les contraintes imposées par le robot ? Tout d'abord, il sera difficilement envisageable d'estimer la capacité de généralisation d'un individu sur un grand ensemble de contextes dans l'environnement réel, et ce même pour un nombre très restreint d'individus. D'autre part, le problème du passage de la simulation à la réalité va également se poser. En conséquence, on peut naïvement se demander s'il est possible de définir une approche associant à la fois transférabilité et généralisation.

Bien que les buts de l'approche par transférabilité et de l'approche *ProGAb* soient évidemment différents, il est intéressant de noter que dans les deux cas on souhaite rendre les solutions aussi peu dépendantes que possible des spécifités du contexte d'évaluation en simulation, que ce soit par rapport au robot physique ou par rapport à d'autres contextes d'évaluation. En se basant sur cette interprétation, les critères de transférabilité et de capacité de généralisation ne semblent pas être conflictuels. Il serait donc possible de définir en premier lieu une approche relativement naïve optimisant 4 critères :

1. la performance en simulation sur un contexte d'apprentissage ;
2. la transférabilité approchée de ce contexte simulé vers le même contexte implémenté en réalité ;
3. la capacité de généralisation approchée sur un ensemble de contextes de validation en simulation ;
4. un critère de diversité comportementale.

Optimiser ces critères revient à chercher une solution performante sur le contexte d'apprentissage, se transférant bien de la simulation à la réalité sur le contexte d'apprentissage et généralisant bien sur d'autres contextes en simulation (cf. illustration sur la figure 5.10). Cette solution va-t-elle obligatoirement être performante sur les contextes de validation implémentés en réalité ? Comme elle est *a priori* peu dépendante des spécifités du contexte d'apprentissage par rapport au dispositif expérimental réel, son transfert sur les contextes de validation en réalité ne devrait échouer que si ces contextes contiennent en simulation des spécifités non présentes dans le contexte d'apprentissage, mais tout de même exploitées par la solution. Or, en supposant que cette solution est performante sur le contexte d'apprentissage en simulation (ce critère étant le plus simple à optimiser), il semble difficile d'imaginer qu'elle est en fait très dépendante de spécifités de la simulation non présentes

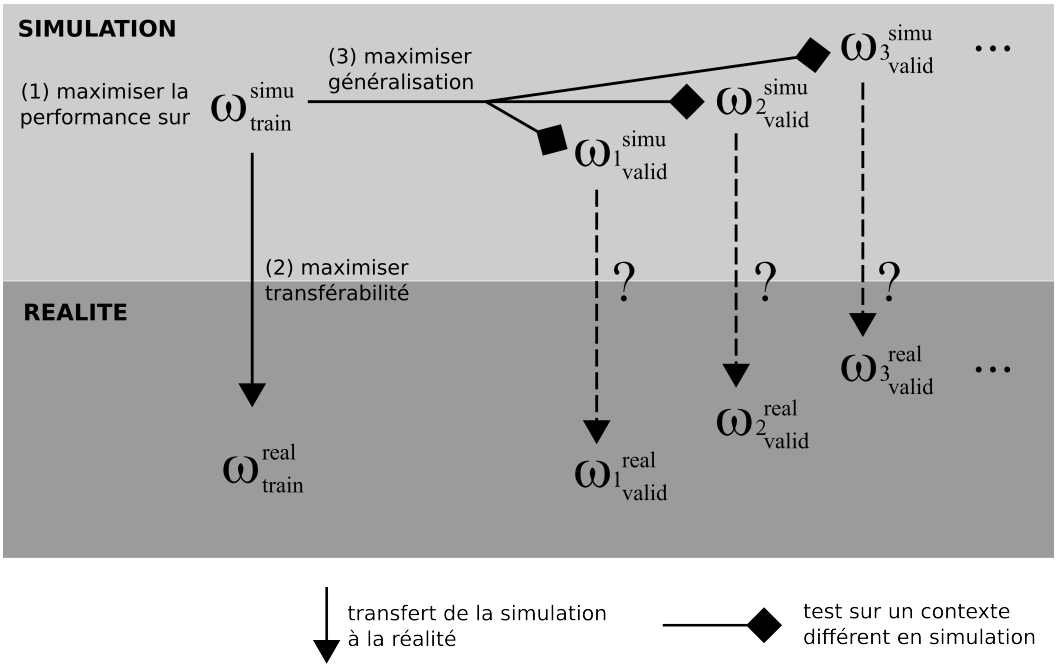

FIGURE 5.10 – Proposition d'approche permettant d'obtenir des contrôleurs pouvant généraliser en réalité : les solutions obtenues vont-elles toujours généraliser sur les contextes de validation en réalité ?

dans ce contexte. Néanmoins, l'utilisabilité de cette approche devra être démontrée empiriquement.

5.8 Conclusion

Nous nous sommes intéressés dans ce chapitre à comment optimiser des contrôleurs possédant des capacités de généralisation. Dans cette optique, nous proposons l'approche *ProGAb* basée sur l'optimisation des solutions sur deux ensembles de contexte, l'un d'apprentissage et l'autre de validation, tout en amoindrissant le nombre d'évaluations par l'utilisation d'un modèle de substitution. Il s'avère que cette approche simple obtient de bons résultats sur deux applications robotiques testées en simulation comparée à des approches classiques habituellement utilisées.

Chapitre 6

S'adapter à partir d'un modèle de soi

6.1 Contexte

Permettre à un robot de se déplacer efficacement dans des environnements et des situations non maîtrisés ou imprévus est un défi majeur en robotique mobile. Considérons par exemple le cas spécifique des robots à locomotion roues-pattes [Endo & Hirose 2000, Grand *et al.* 2004b, Jarrault *et al.* 2010] (cf. figure 6.1). Ces robots visent à combiner l'efficacité des robots à roues avec la versatilité des robots marcheurs et ils peuvent potentiellement s'adapter à de nombreux types de terrains du fait de leurs nombreux modes de locomotion possibles (marcher, rouler, ramper, grimper, ...). Néanmoins, même si certains travaux proposent des contrôleurs pour implémenter un mode de locomotion donné [Endo & Hirose 2000, Grand *et al.* 2004b, Jarrault *et al.* 2010], l'adaptation de locomotion de ces robots hybrides à un nouvel environnement ou une nouvelle situation reste une question ouverte.

Par exemple, supposons que le robot se déplace initialement sur une surface plane en roulant. Si le robot arrive ensuite dans un terrain recouvert d'herbe, son comportement de roulement ne sera plus performant. Un mécanisme d'adaptation est alors nécessaire pour trouver rapidement un nouveau comportement lui permettant de se déplacer correctement sur l'herbe. La littérature propose plusieurs processus pour permettre au robot de s'adapter : soit le nouveau comportement est choisi dans une base de comportements pré-appris (e.g. [Steingrube *et al.* 2010]), soit le

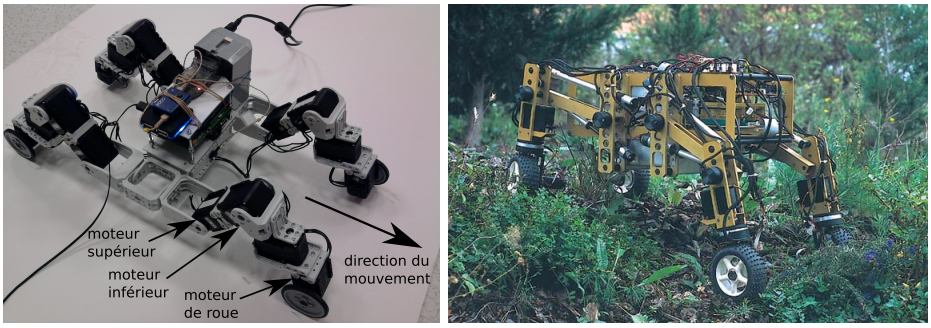

Figure 6.1 – A gauche, robot quadrupède à locomotion hybride roues-pattes utilisé dans les expériences (12 degrés de liberté), construit d'après le robot Hylos (à droite).

robot découvre un comportement intéressant par optimisation ou par apprentissage (e.g. [Kohl & Stone 2004, Bongard *et al.* 2006, Christensen *et al.* 2010]). Dans le premier cas, le robot ne découvre rien à proprement parler : il opère juste une sélection entre plusieurs comportements possibles déjà connus. Si aucun de ces comportements n'est adapté à la nouvelle situation rencontrée, une telle méthodologie ne pourra pas permettre au robot de s'adapter correctement. *A contrario*, lancer une optimisation ou un apprentissage dans la nouvelle situation peut permettre au robot de trouver des comportements originaux, et ce sans que son concepteur n'ait à faire un inventaire des situations que le robot est susceptible de rencontrer. Pour cette raison, le processus de découverte que nous proposons ici repose sur une méthode d'optimisation.

Dans la suite de ce chapitre, nous nous plaçons dans le cadre de phases d'adaptation discrètes : tant que le robot est dans une situation donnée, il peut utiliser le même comportement sans variations importantes de performance. Par exemple, dans une première situation, le robot se déplace en roulant sur un sol plat sur plusieurs mètres. Le robot entre ensuite dans un champ d'herbe, rencontrant donc une situation différente. L'adaptation qui va nous intéresser est celle nécessaire pour découvrir un nouveau comportement efficace à son entrée dans le champ d'herbe. Dans ce contexte, un mécanisme d'adaptation faisant intervenir un processus de découverte par optimisation peut être découpé en 3 différentes phases comme illustré sur la figure 6.2. Ce découpage s'inspire en partie des travaux présentés dans l'article [Bongard *et al.* 2006] :

1. Le robot détecte qu'il a changé de situation, par exemple que son environnement a changé ou que son comportement n'est plus performant.
2. Le robot lance un processus de découverte afin de trouver un nouveau mode de locomotion efficace dans la nouvelle situation.
3. Un comportement de locomotion performant est ensuite sélectionné et exploité, et ce tant que le robot ne détecte pas un changement de situation.

Dans un tel mécanisme d'adaptation, le processus de découverte doit permettre au robot de trouver un nouveau contrôleur performant à utiliser dans la nouvelle situation. D'après la littérature (cf. partie 2.4.3), ce processus peut être assuré via une phase d'optimisation ou d'apprentissage opérée directement sur le robot (e.g. [Kohl & Stone 2004, Sproewitz *et al.* 2008]). L'optimisation prendra alors directement en compte la nouvelle situation dans laquelle se trouve le robot sans qu'il soit nécessaire d'indiquer explicitement les changements intervenus par rapport à la situation précédente. Le nombre d'évaluations nécessaires peut néanmoins être important et le mécanisme d'adaptation obtenu peut être assez lent : par exemple, 3 heures pour apprendre un comportement de marche quadrupède sur un robot Sony Aibo [Kohl & Stone 2004] ou environ 20 minutes dans le cas d'un robot serpent [Sproewitz *et al.* 2008].

Afin que le robot puisse s'adapter à tout type de situation sans avoir à faire un trop grand nombre d'évaluations, certains travaux proposent un mécanisme alternatif où le processus de découverte s'appuie sur un modèle de soi (ou *self-model*)

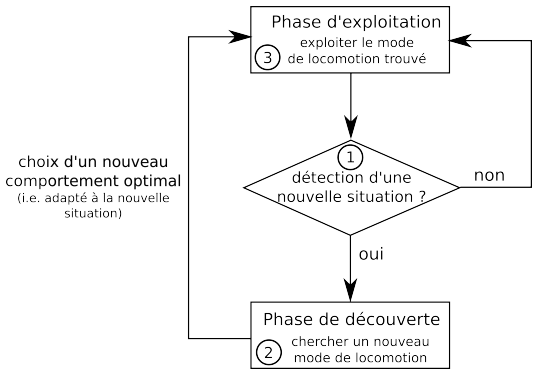

FIGURE 6.2 – Schéma d'un mécanisme d'adaptation de locomotion. Le robot détecte une nouvelle situation (étape 1). Il lance ensuite un processus de découverte pour trouver un nouveau mode de locomotion (étape 2). Quand un contrôleur adapté à la nouvelle situation est trouvé, il est exploité jusqu'à la détection d'une nouvelle situation (étape 3).

du robot qu'il construit dans chaque nouvelle situation à l'aide de quelques expériences sur lui-même [Bongard *et al.* 2006, Bongard 2009]. Une fois qu'un modèle suffisamment précis est obtenu, il peut être utilisé comme simulateur pour trouver un comportement adapté à la situation rencontrée [Bongard *et al.* 2006]. A chaque nouvelle situation, i.e. quand une incohérence entre les prédictions du modèle de soi et le comportement effectif du robot physique est détectée, le robot identifie un nouveau modèle à partir de données générées en réalité. Le processus de découverte implique alors peu d'expériences sur le robot : quelques expériences sont effectuées pour construire le modèle de soi dans chaque nouvelle situation et l'optimisation a lieu uniquement sur ce modèle. Le mécanisme d'adaptation est potentiellement plus rapide qu'avec une optimisation directe en réalité. Cette approche a notamment été appliquée dans le cadre d'une expérience de robotique résiliente [Bongard *et al.* 2006] où un robot quadrupède doit s'adapter à un changement de morphologie. Après la perte d'une de ses pattes, le robot construit un nouveau modèle de soi où la nouvelle morphologie est prise en compte en effectuant 15 expériences en réalité. Il peut ensuite lancer une optimisation sur ce modèle et découvrir des comportements de locomotion originaux pour se mouvoir à nouveau efficacement.

Un processus de découverte cherchant à construire ou améliorer le modèle de soi du robot à chaque nouvelle situation peut s'accompagner de deux inconvénients importants que nous allons détailler :

- La construction d'un modèle de soi est un processus potentiellement complexe d'identification de modèles et pouvant nécessiter un grand nombre de données expérimentales sur le robot.
- En optimisant le contrôleur uniquement à partir du modèle de soi, l'approche n'échappe pas aux problèmes de passage à la réalité.

Si le mécanisme d'adaptation repose sur la construction d'un modèle de soi dans chaque situation, le robot a deux manières de construire le nou-

veau modèle : en partant de zéro ou en améliorant son modèle de soi précédent. Dans les deux cas, la difficulté de l'étape d'identification dépend de l'information *a priori* dont on dispose sur la structure du modèle. Si la structure du modèle est déjà connue et qu'on cherche à fixer la valeur de quelques paramètres, le coût expérimental nécessaire pour une identification correcte du modèle peut être assez faible [Bongard & Lipson 2005, Abbeel *et al.* 2006, Abbeel *et al.* 2007, Zagal *et al.* 2009]. Dans le cas général si la structure du modèle n'est pas connue, le processus d'identification revient à trouver une structure, que ce soit par un modèle mathématique (e.g. [Khosla & Kanade 1985, Kozlowski 1998, Albu-Schaffer & Hirzinger 2001]) ou une simulation dynamique (e.g. [Joukhadar *et al.* 1997]), avant d'identifier les valeurs des paramètres libres. La structure trouvée doit aussi être suffisamment flexible pour modéliser le robot dans toute nouvelle situation. Par exemple si le robot perd une patte, la structure du modèle doit autoriser le fait que cette patte ne soit plus prise en compte ou que sa masse devienne négligeable [Bongard *et al.* 2006]. Suivant le même principe, si le robot marche sur le sol dans un premier temps, puis doit ensuite voler jusqu'à une destination donnée, la structure du modèle de soi doit pouvoir intégrer des mécanismes de la mécanique des fluides qui ne sont cependant pas utiles quand le robot est à terre. Si la structure ne peut pas prendre en compte la mécanique des fluides, chercher un modèle de soi dans la situation où le robot doit voler reviendrait à redécouvrir les lois de l'aérodynamique. En résumé, l'étape de construction d'un modèle de soi peut être complexe et coûteuse, notamment en temps expérimental, ce qui ne garantit pas un processus d'adaptation rapide.

Ce processus de découverte ne permet pas non plus d'échapper aux problèmes de passage à la réalité. En effet, une fois qu'un modèle de soi est obtenu, il est utilisé comme simulateur pour découvrir un contrôleur adapté à la nouvelle situation rencontrée par le robot. L'optimisation finale du contrôleur ne se faisant que sur ce modèle nouvellement construit, il est donc possible que le meilleur contrôleur trouvé en simulation ne soit pas aussi efficace sur le robot. De telles pertes de performance ont d'ailleurs été observées dans l'expérience de robotique résiliente citée plus haut [Bongard *et al.* 2006]. On peut d'ailleurs remarquer que plus le robot cherchera à construire un modèle de soi fiable pour diminuer ces problèmes de passage à la réalité, plus le nombre d'expériences à effectuer sera important et moins le mécanisme d'adaptation sera rapide.

L'approche alternative que nous proposons part de l'hypothèse que, dans certains cas, le modèle de soi déjà construit par le robot peut être encore fiable dans une nouvelle situation. Par exemple, en supposant qu'un robot pouvant rouler ou marcher utilise comme modèle de soi un simulateur physique modélisant sa locomotion sur un sol plat. Si le robot se déplace dans l'herbe où il ne peut pas rouler efficacement, les prédictions de son modèle de soi vont diverger avec la réalité pour les comportements de roulement. En comparaison, certains comportements de marche vont être relativement bien simulés. Le robot peut donc dans ce cas utiliser son modèle de soi afin de trouver un comportement de marche à la fois efficace et relativement bien simulé, i.e. transférable en réalité. Quelle que soit la fiabilité du modèle de soi, s'il

est possible de l'utiliser comme simulateur pour trouver des comportements performants et transférables sur le robot, alors il est possible de trouver des contrôleurs adaptés à la nouvelle situation.

Nous proposons de reformuler ce problème d'adaptation en un problème de transférabilité entre le modèle de soi et le robot dans une situation donnée. Dans ce contexte, le modèle de soi va rester fixe quelle que soit la nouvelle situation rencontrée par le robot et il sera utilisé comme un simulateur pour trouver des contrôleurs à la fois efficaces et transférables sur le robot via l'approche par transférabilité présentée auparavant dans le chapitre 4. Un tel processus de découverte possède les avantages suivantes :

- On ne cherche pas à déterminer les changements entre la nouvelle situation et la précédente. On cherche seulement les parties du modèle de soi performantes et valides pour modéliser la situation rencontrée par le robot.
- L'approche prend explicitement en compte les problèmes du passage à la réalité en cherchant des contrôleurs transférables.
- Le processus de découverte nécessite peu d'expériences sur le robot, étant donné que l'optimisation se fait essentiellement sur le modèle de soi et qu'on ne cherche pas à améliorer ou re-construire ce modèle à chaque situation. Ce coût expérimental amoindri devrait donc permettre un mécanisme d'adaptation rapide.

Il est cependant important de noter que les performances de cette approche dépendent essentiellement de la capacité du modèle de soi du robot à rester relativement fiable dans les nouvelles situations, c'est-à-dire s'il existe ou non des solutions efficaces d'après le modèle et transférables sur le robot. Ce point concernant l'applicabilité de l'approche sera discuté plus en détails dans la partie 6.4.2.

6.2 Processus de découverte via l'approche par transférabilité

6.2.1 Principes de l'algorithme

Nous supposons en premier lieu que le robot a déjà construit son modèle de soi. Ce modèle reste fixe quelle que soit la situation rencontrée par le robot. Le processus de découverte utilise l'approche par transférabilité (cf. partie 4) directement sur ce modèle de soi afin de trouver des contrôleurs à la fois performants d'après le modèle et transférables sur le robot dans la situation courante. Le processus de découverte proposé utilise un algorithme évolutionniste multi-objectif pour optimiser les solutions et va vérifier les points suivants :

- La transférabilité exacte des contrôleurs est estimée par une mesure de disparité entre simulation et réalité (disparité $S|R$) notée D^* calculée pour un contrôleur en comparant le comportement correspondant prédit du modèle de soi et celui observé sur le robot.
- Le modèle de substitution $\hat{D}$ utilisé comme approximation de la mesure de

disparité $S|R$ est interpolé par IDW [Shepard 1968] à partir des valeurs exactes de disparité $S|R$ correspondant aux contrôleurs déjà transférés sur le robot.

- Ce modèle de substitution est construit sur un espace comportemental : il prend en entrée une description comportementale des solutions optimisées.
- Pour un contrôleur c donné, la valeur prédite de transférabilité correspondante dépend de la proximité comportementale entre c et les contrôleurs déjà transférés selon une fonction de distance ou métrique *in silico* notée b_{dist}.
- Le choix du prochain contrôleur à transférer se fait suivant la stratégie de mise à jour du modèle de substitution et dépend de la distance aux contrôleurs déjà transférés.
- En supposant que n objectifs de performance doivent être maximisés sur la tâche considérée, les contrôleurs de la population sont évalués par les $n + 2$ objectifs suivants :
  1. les n objectifs de performance systématiquement évalués dans le modèle de soi ;
  2. la disparité $S|R$ approchée obtenue à l'aide du modèle de substitution ;
  3. un objectif de diversité comportementale comme défini dans la partie 4.2.2.

L'algorithme d'optimisation utilisé suit les mêmes étapes que celui présenté dans la partie 4.2.4 (voir schéma de la figure 4.5). Seule la stratégie de mise à jour du modèle de substitution est différente (étape B sur la figure 4.5) : le contrôleur de la population dont la diversité comportementale est maximale est périodiquement transféré sur le robot toutes les N générations. Ce choix permet d'assurer qu'un processus d'optimisation ait bien lieu entre deux expériences de transfert successives.

Une fois que le processus de découverte est terminé, l'algorithme évolutionniste multi-objectif renvoie l'ensemble des solutions non-dominées en fonction des différents objectifs (n + 2, si n est le nombre d'objectifs de performance définis sur la tâche). Il faut ensuite choisir la meilleure solution parmi ces différents compromis qui sera ensuite exploitée par le robot jusqu'au prochain changement de situation.

6.2.2 Meilleure solution de l'optimisation

Nous supposons tout d'abord qu'un seuil $D^*_{threshold}$ peut être empiriquement défini sur les valeurs de disparité $S|R$ de telle façon que des valeurs de disparité inférieures que $D^*_{threshold}$ correspondent à des transferts de bonne qualité. Il s'agit d'un critère fourni par l'utilisateur qui indique à partir de quelle valeur les comportements en réalité et en simulation sont considérés comme proches.

A la fin du processus d'évolution, l'algorithme multi-objectif renvoie un ensemble de solutions non-dominées $\mathcal{E}$. Nous définissons alors l'ensemble transférable des solutions non-dominées $\mathcal{T}$ comme le sous-ensemble de $\mathcal{E}$ contenant les solutions non-dominées dont la valeur de disparité $S|R$ est inférieure au seuil $D^*_{threshold}$. Deux cas sont alors envisageables : si cet ensemble est vide, le meilleur compromis sélectionné est la solution non-dominée de disparité $S|R$ minimale, bien qu'elle ne devrait pas se transférer correctement. Si l'ensemble n'est pas vide, nous avons à choisir une solution compromis dans cet ensemble transférable $\mathcal{T}$. Le choix du meilleur compromis

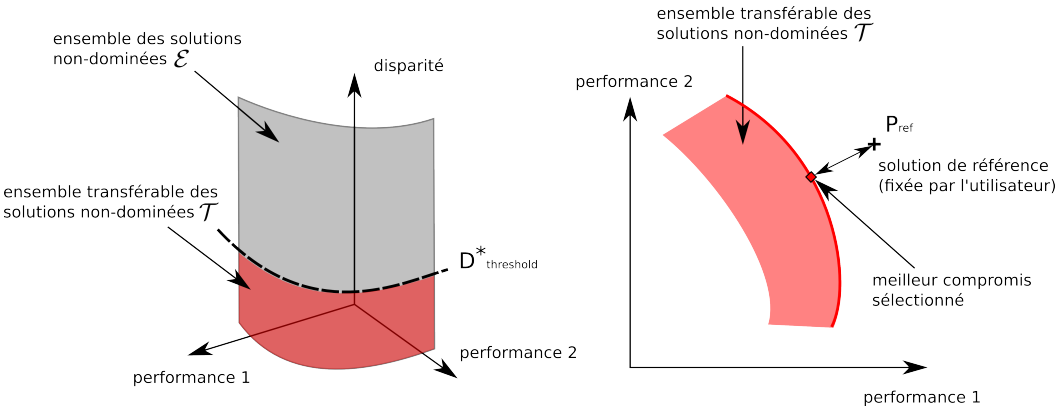

FIGURE 6.3 – Sélection de la meilleure solution. A gauche : l'ensemble des solutions non-dominées trouvées par l'algorithme évolutionniste multi-objectif peut être découpé en deux parties : les solutions transférables et les solutions non-transférables. A droite : parmi les solutions transférables, on sélectionne comme meilleur compromis le contrôleur dont la distance euclidienne à un point de référence P_{ref} est minimale.

dans cet ensemble se fait à l'aide d'un point de référence P_{ref} dans l'espace des objectifs : ce point correspond à des valeurs idéales pour chacun des objectifs définies par l'utilisateur. On sélectionne alors comme meilleur compromis le point de l'ensemble transférable des solutions non-dominées qui minimise la distance euclidienne à P_{ref}.

Comme indiqué dans la partie 6.2.2, l'approche par transférabilité originale utilise également un point de référence, le point idéal transférable (cf. figure 6.3), défini à partir de l'ensemble transférable des solutions non-dominées $\mathcal{T}$: le point idéal transférable peut varier d'une optimisation à l'autre. Ici, nous préférons nous baser sur les connaissances de l'utilisateur pour définir un point de référence fixe.

L'ensemble du processus de choix de la meilleure solution est représenté sur la figure 6.3.

6.3 Application à la locomotion hybride roues-pattes

Les résultats présentés dans cette section ont fait l'objet d'une publication à la conférence *CLAWAR* en 2011 dans l'article [Koos & Mouret 2011].

6.3.1 Contexte

Les robots à locomotion hybride roues-pattes, par exemple les deux robots quadrupèdes de la figure 6.1, sont conçus pour pouvoir à la fois exploiter l'efficacité des robots à roue et la versatilité des robots à pattes [Endo & Hirose 2000, Grand *et al.* 2004b, Jarrault *et al.* 2010]. Leurs pattes se terminant par des roues,

ils peuvent en effet rouler sur des terrains simples et adapter leur posture sur des sols irréguliers ; ils peuvent également bloquer leurs roues et utiliser des comportements de marche. Plusieurs travaux traitent du contrôle de ces robots [Endo & Hirose 2000, Grand *et al.* 2004b, Jarrault *et al.* 2010] pour proposer par exemple un contrôleur tirant avantage des articulations des pattes pour ajuster la posture du robot [Grand *et al.* 2004b, Jarrault *et al.* 2010]. En comparaison, peu de travaux se sont intéressés à trouver différents modes de locomotion, comme par exemple rouler avec des roues non motrices tout en marchant [Endo & Hirose 2000]. Néanmoins, une question importante pour ce type de robots reste ouverte : comment le robot peut-il découvrir un nouveau contrôleur adapté dans une nouvelle situation ? L'approche que nous introduisons dans ce chapitre et appliquant l'approche par transférabilité à un modèle de soi du robot a initialement été proposée pour apporter une réponse à ce problème [Koos & Mouret 2011].

Les différents modes de locomotion que peuvent utiliser les robots à locomotion hybride roues-pattes (rouler, marcher, ramper, grimper, ...) peuvent être interprétés comme différents compromis entre 2 objectifs de performance : l'efficacité de la locomotion et l'énergie dépensée par le robot pour accomplir son comportement. Par exemple, marcher consomme plus d'énergie que rouler, mais est aussi plus versatile, étant donné qu'un comportement de marche peut fonctionner sur des terrains plats et dans l'herbe.

Dans ce contexte, l'algorithme de découverte va optimiser 4 objectifs :

- la vitesse moyenne du robot d'après le modèle de soi ;
- l'énergie dépensée par le robot d'après le modèle de soi ;
- la valeur de disparité $S|R$ approchée d'après le modèle de substitution ;
- la valeur de diversité comportementale.

6.3.2 Dispositif expérimental

Le robot quadrupède à locomotion hybride roues-pattes considéré ici a été construit à l'aide d'un Kit de Bioloid d'après le robot Hylos [Grand *et al.* 2004a] (voir figure 6.1), initialement conçu pour l'exploration autonome de terrains accidentés (volcanique par exemple). Chaque patte du robot est terminée par une roue et le contrôle va porter sur les 2 moteurs d'une patte les plus proches du corps du robot et le moteur de la roue, soit 12 degrés de liberté au total. Une patte se compose donc d'un moteur supérieur, d'un moteur inférieur et d'un moteur de roue comme indiqué sur la partie gauche de la figure 6.1.

Le contrôle du robot dépend de 3 paramètres réels $(p_1,\ p_2,\ p_3) \in [-1,\ 1]^3$: p_1 définit la posture du robot, p_2 l'amplitude des mouvements des pattes and p_3 la vitesse des roues. L'effet du paramètre p_1 est illustré sur la figure 6.4. Les caractéristiques comportementales b^1, b^2, b^3 intervenant dans le calcul de la métrique *in silico* b_{dist} (cf. partie 4.1.3) sont directement obtenues à partir de ces paramètres de contrôle. La correspondance est indiquée dans les équations ci-dessous. Concernant le contrôleur des moteurs inférieur et supérieur de chaque patte, la position angulaire désirée du moteur i au temps t est calculée comme suit :

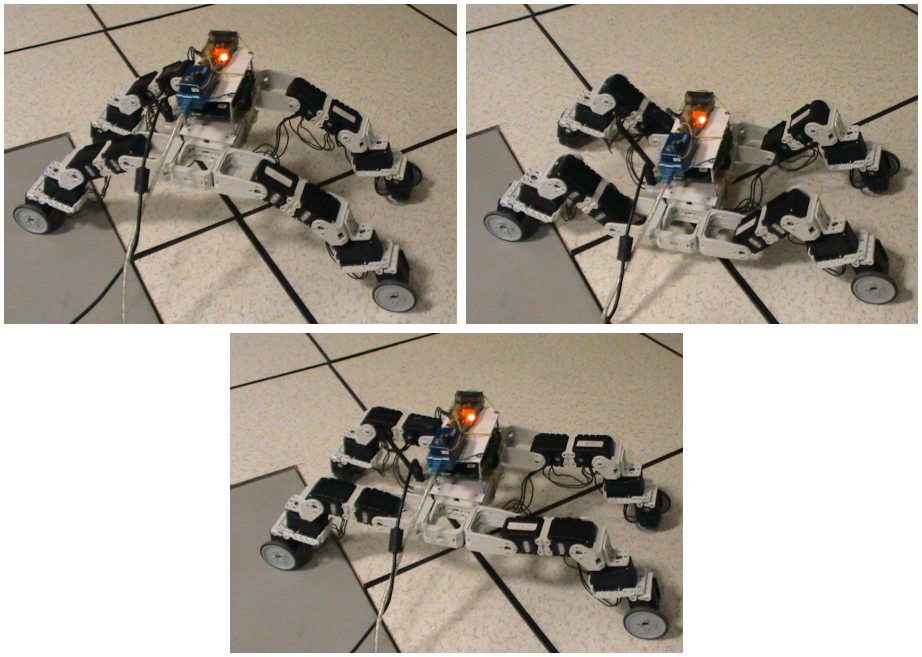

FIGURE 6.4 – Posture du robot en fonction du paramètre de contrôle p_1. A gauche, corps surélevé quand $p_1 < 0$. A droite, corps abaissé quand $p_1 > 0$. En bas, corps horizontal quand $p_1 = 0$.

$$\alpha^d(i,t) = \frac{5\pi}{48} \cdot p_1 - \begin{cases} \frac{5\pi}{24} \cdot p_2 \cdot sin(2\pi t - \phi(i)) & \text{, si } p_2 > 0 \quad (b^1 = p_1,\ b^2 = p_2) \\ 0 & \text{, sinon} \qquad\quad (b^1 = p_1,\ b^2 = 0) \end{cases}$$

La phase $\phi(i)$ vaut 0 pour le moteur supérieur de la patte et $\frac{\pi}{2}$ pour le moteur inférieur de la patte. Ces deux moteurs sont donc contrôlés par le même signal mais avec des phases différentes. De nouvelles positions angulaires sont envoyées toutes les 0.05 secondes.

Les moteurs des 4 roues sont contrôlés avec la même valeur de vitesse v fixe qui ne dépend que du paramètre p_3 ($v_{max} = 6$ rad/s) :

$$v = \begin{cases} p_3 \cdot v_{max} & \text{, si } p_3 > 0 \quad (b^3 = p_3) \\ 0 & \text{, sinon} \qquad (b^3 = 0) \end{cases}$$

Le modèle de soi utilisé est une représentation du robot dans le simulateur dynamique Open Dynamics Engine [Smith 2005] (ODE). Le modèle ODE se compose de 14 corps rigides, 8 moteurs angulaires (*angular motors*) correspondant aux moteurs inférieurs et supérieurs, ainsi que 4 articulations (*hinge*) pour les moteurs des roues. Quel que soit l'environnement réel du robot, le modèle de soi est toujours utilisé pour simuler la locomotion du robot sur un sol plat pendant une seconde et ainsi évaluer sa vitesse moyenne et l'énergie dépensée. L'estimation de l'énergie dépensée se fait en sommant les mouvements angulaires de chaque degré de liberté.

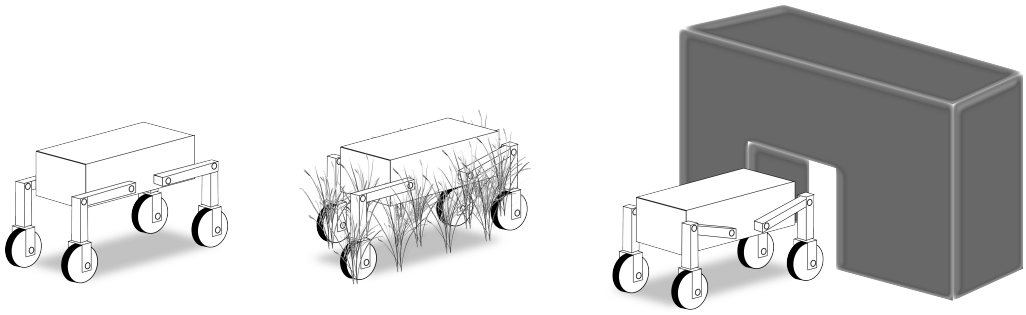

FIGURE 6.5 – Les trois types de situations considérées dans le dispositif expérimental : (gauche) sol plat uniforme, (centre) herbe (le robot ne peut pas utiliser ses roues), (droite) tunnel (le robot doit se baisser afin d'entrer dans le tunnel, sinon ses mouvements sont bloqués).

Quand un contrôleur est transféré sur le robot, la vitesse moyenne du robot est mesurée sur une seconde en utilisant deux scanners CODA cx1 (Charnwood Dynamics Ltd, UK) pour détecter la position absolue d'un marqueur posé sur le corps du robot. La fonction de disparité $S|R$ exacte est alors définie comme la variation absolue entre la vitesse moyenne sur le robot et la vitesse moyenne en simulation correspondant au contrôleur évalué.

Les 4 objectifs sont optimisés simultanément par l'algorithme évolutionniste multi-objectif NSGA-II [Deb *et al.* 2002b] (cf. partie 2.2.6.2 pour plus de détails sur cet algorithme), avec une population de 40 individus pendant 200 générations. Le génotype d'un contrôleur est le vecteur des 3 paramètres de contrôle $(p_1,\ p_2,\ p_3) \in [-1,\ 1]^3$. Deux opérateurs sont définis sur ce génotype : une mutation ponctuelle gaussienne de moyenne nulle et d'écart-type 0.2 et un opérateur de croisement. La probabilité de mutation est de 0.15.

Le modèle de substitution de la fonction de disparité $S|R$ est mis à jour toutes les 10 générations ($N = 10$), en transférant le contrôleur de la population dont la valeur de diversité comportementale est maximale. Une phase de découverte complète s'accompagne donc de 20 expériences en réalité, chacune durant 1 seconde, soit environ 20 secondes de temps expérimental pendant l'optimisation.

Le point de référence P_{ref} utilisé pour sélectionner le meilleur compromis à la fin de l'optimisation correspond à une vitesse moyenne de 18 cm/s and une valeur d'énergie dépensée de 25. Ces valeurs correspondent au comportement de roulement le plus efficace prédit par le modèle de soi du robot sur un sol plat.

Le processus de comportement est testé dans 3 situations où le robot doit s'adapter comme indiqué sur la figure 6.5 :

T1. *sol plat* — pas de contrainte particulière ;

T2. *terrain herbeux* — les roues du robot sont bloquées ;

T3. *tunnel* — les mouvements du robot sont bloqués si $p_1 < 0.75$, i.e. si le corps du robot n'est pas suffisamment bas.

TABLE 6.1 – Vitesse moyenne en réalité (cm/s) et énergie dépensée en simulation (unité arbitraire) sur les trois différents terrains pour les meilleurs contrôleurs obtenus dans chaque terrain (5 répétitions, valeurs d'écart-type entre parenthèses).

	sol plat	herbe	tunnel	énergie
optim. sur le sol plat	17.1 (0.3)	0.0 (0)	8.6 (9.9)	26 (3)
optim. sur l'herbe	13.1 (1.9)	13.1 (1.9)	8.3 (7.6)	33 (1)
optim. dans le tunnel	15.9 (1.7)	0.0 (0)	15.9 (1.7)	27 (2)

6.3.3 Résultats

L'algorithme de découverte proposé est testé 5 fois dans chacune des 3 situations décrites ci-dessus. Les résultats obtenus sont indiqués dans la table 6.1. Les comportements typiques obtenus dans chacune des situations sont indiqués sur la figure 6.6. D'un point de vue qualitatif, nous observons que :

- le robot sélectionne systématiquement un comportement de roulement sur le terrain plat (figure 6.6, gauche) ;
- le robot découvre un comportement de marche dans le terrain herbeux (figure 6.6, centre) quand ses roues ne sont plus utilisables ;
- le robot abaisse son corps pour entrer dans le tunnel et sélectionne un comportement de roulement (figure 6.6, droite) ;

Afin d'illustrer l'intérêt qu'apporte l'adaptation de locomotion dans cette expérience simple, la table 6.2 indique la vitesse moyenne des comportements découverts dans chacune des situations quand ils sont testés dans l'ensemble des 3 situations. Il s'avère qu'en s'adaptant à chaque nouvel environnement, le robot obtient une vitesse moyenne de 15.4 cm/s (sd = 0.5), ce qui est significativement meilleur que s'il utilise un comportement fixe découvert dans une des trois situations (p-values $< 8.4 \cdot 10^{-3}$ d'après un test de Welch). D'autre part, la phase de découverte dure environ 2 minutes : environ 20 secondes de temps expérimental et le reste pour l'optimisation multi-objectif avec un ordinateur multi-cœur récent [1]. Le processus de découverte proposé permet au robot de s'adapter rapidement aux nouvelles situations. De plus, comme la majeure partie du processus se déroule en simulation, la durée de l'optimisation peut encore être réduite en utilisant un ordinateur plus puissant contrairement à une optimisation directe sur le robot par exemple.

6.4 Discussion

6.4.1 Vers une mesure de transférabilité embarquée

Dans l'application considérée ici ainsi que dans les deux dispositifs expérimentaux présentés dans le chapitre 4, le calcul de la mesure de disparité $S|R$ implique de pouvoir obtenir le comportement du robot physique : la trajectoire dans les

1. Intel Xeon E5520 dual quad-core à 2.3 GHz.

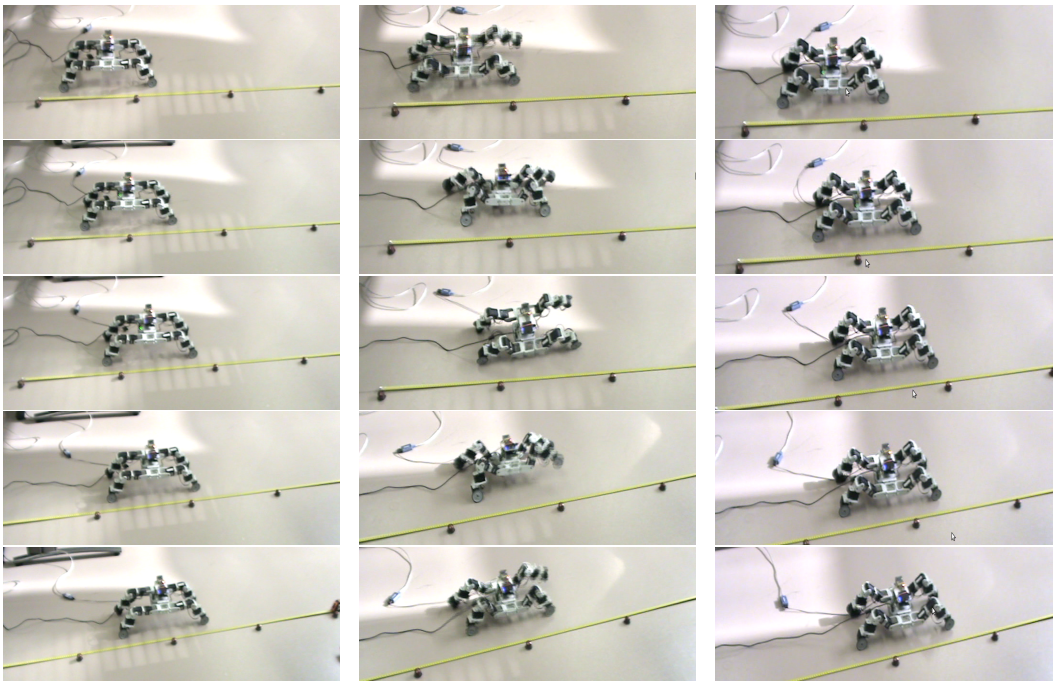

FIGURE 6.6 – Comportements typiques sélectionnés dans chaque terrain : (gauche) comportement de roulement sur T1 (sol plat) ; (centre) marche sur T2 (terrain herbeux) ; (droite) comportement de roulement avec un corps abaissé sur T3 (tunnel).

TABLE 6.2 – Vitesse moyenne (cm/s) sur l'ensemble des trois situations des contrôleurs trouvés dans une situation donnée (écart-types entre parenthèses). La dernière ligne indique la vitesse moyenne obtenue avec l'algorithme de découverte proposé, i.e. en s'adaptant à chaque nouvelle situation. En s'adaptant, le robot se déplace significativement plus vite qu'en utilisant un comportement fixe (p-value $< 8.4 \cdot 10^{-3}$ d'après un test de Welch).

	vitesse moyenne sur les 3 situations
optim. sur le sol plat	8.6 (3.3)
optim. sur l'herbe	11.5 (1.5)
optim. dans le tunnel	10.6 (1.1)
avec adaptation	15.4 (0.5)

expériences du chapitre 4 et la distance parcourue ici. Cette information est obtenue dans les deux cas de manière externe à l'aide d'un processus de capture de mouvement utilisant des scanners cx1 CODA. Il est évident que le recours à un tel équipement limite dans une certaine mesure l'applicabilité de l'approche à des robots plus volumineux ou à des environnements extérieurs. Afin de résoudre ce problème, une possibilité revient à se baser uniquement sur des informations internes sensori-motrices accessibles par le robot pour calculer la disparité $S|R$.

Des résultats récents indiquent qu'il est possible de comparer des comportements robotiques en simulation sur la base des séries temporelles des capteurs [Doncieux & Mouret 2010]. Néanmoins, dans le cadre d'un calcul de disparité $S|R$, cela implique que les valeurs des capteurs soient modélisées avec précision dans le simulateur, malgré parfois de hauts niveaux de bruit et des capteurs peu fiables. Il est parfois difficile d'effectuer des comparaisons précises entre deux séries temporelles de capteurs, car de faibles disparités initiales peuvent rapidement s'accumuler et mener à des signaux très différents [Bongard & Lipson 2005], d'où la préférence de mesures externes pour comparer le comportement réel et le comportement simulé d'un robot [Bongard *et al.* 2006].

Une voie prometteuse alternative est d'utiliser ces informations sensori-motrices pour dériver une estimation de la trajectoire du robot par intégration de capteurs [Borenstein *et al.* 1997, Barshan & Durrant-Whyte 1995]. Néanmoins, les méthodes d'intégration s'accompagnent d'un phénomène de dérive entre la trajectoire estimée et la trajectoire réelle ce qui nécessite de recaler périodiquement l'estimation pour éviter une accumulation des déviations. Plus récemment, des résults intéressants ont été obtenus sur des robots à roue utilisant de l'odométrie visuelle embarquée [Nistér *et al.* 2006, Campbell *et al.* 2005] ou des techniques de SLAM (Simultaneous Localization and Mapping) [Dissanayake *et al.* 2001, Kleiner *et al.* 2006], permettant d'obtenir des estimations précises de la trajectoire. De telles méthodes utilisent seulement les données d'une unique caméra [Campbell *et al.* 2005], éventuellement couplée avec des informations de capteurs inertiels [Nistér *et al.* 2006] et semblent à la fois robustes et simples à implémenter.

Il semble donc possible de définir une mesure de disparité $S|R$ fiable à partir de comportements obtenus de manière embarquée. Ce point sera étudié en priorité dans nos prochains travaux.

6.4.2 Applicabilité dans le cas de la robotique mobile

Notre approche repose sur l'hypothèse principale que le modèle de soi fixe du robot permet de capter certaines parties de sa dynamique dans les nouvelles situations qu'il rencontre : il est alors possible de trouver des comportements efficaces et transférables à partir de ce simulateur. L'applicabilité de l'approche dépend donc directement de la capacité du modèle de soi du robot d'être encore fiable dans la nouvelle situation rencontrée, qu'il s'agisse d'un changement d'environnement ou d'une modification morphologique suite à une panne.

Si l'environnement varie drastiquement, le modèle de soi du robot peut ne plus

être exploitable du tout. Par exemple, si un robot se déplace initialement sur un sol plat et doit ensuite traverser un lac, un modèle de soi du robot sur le sol plat ne va pas être utilisable dans l'eau. Dans ce contexte, soit le robot doit ré-apprendre un modèle de soi dans le nouvel environnement (e.g. [Bongard *et al.* 2006]), soit il faut intégrer suffisamment d'information sur cet environnement dans la simulation pour la rendre plus fiable. Dans ce second cas, il est par exemple possible d'extraire les caractéristiques de l'environnement à partir des valeurs de capteurs du robot et d'en inférer le type d'environnement via des méthodes statistiques (e.g. [Lenser & Veloso 2004, Giguere *et al.* 2006]). Dans l'exemple du robot-amphibie plongeant dans un lac, il est envisageable que le robot découvre lui-même à l'aide de ses capteurs qu'il se trouve dans l'eau [Giguere *et al.* 2006]. Il peut ensuite prendre en compte cette information quand il simulera sa locomotion à l'aide de son modèle de soi, rendant ainsi son simulateur plus fiable. Cette méthode a l'avantage de ne pas impliquer un processus d'identification complet du modèle de soi et d'éviter un nombre potentiellement important d'expériences sur le robot.

Dans le cas où la morphologie du robot est modifiée suite à une panne et où la dynamique du robot peut drastiquement changer, on peut aussi se demander si l'approche par transférabilité serait applicable sans que le robot ne doive identifier d'abord un nouveau modèle de soi plus fiable. Par exemple, dans l'expérience de robotique résilient présentée dans l'article [Bongard *et al.* 2006], un robot quadrupède perd une partie d'une patte ce qui l'oblige à trouver des comportements compensatoires pour se déplacer à nouveau. Dans ce cas, la modification morphologique entraîne une modification importante de la dynamique du robot : un simulateur fiable pour modéliser le robot quadrupède initial ne pourrait sans doute pas bien simuler des locomotions tripèdes après la perte de la patte. Certains travaux en robotique proposent de construire des robots avec une certaine redondance de leurs degrés de liberté afin de les rendre entre autre plus robustes à d'éventuelles pannes [Ito & Gofuku 2003, Yang 2005]. Dans le cas particulier de la robotique mobile ou d'exploration, cela peut signifier qu'un robot doive posséder un nombre important de pattes [Wettergreen & Thorpe 1992]. Par exemple, en cas de panne sur une patte, un robot octopode aura toujours 7 pattes à disposition pour assurer un comportement efficace et sa dynamique avec 7 pattes peut rester potentiellement proche de sa dynamique initiale. L'approche par transférabilité pourrait donc être utilisé sur ce type de robots redondants, et ce malgré d'éventuelles pannes impliquant des changements morphologiques.

Malgré l'hypothèse contraignante que le modèle de soi du robot doive rester fiable dans les nouvelles situations rencontrées, le mécanisme d'adaptation proposé ici semble également applicable dans certains cas où des changements drastiques de l'environnement du robot ou de sa morphologie sont observés.

6.4.3 Validation de l'approche

Les résultats présentés dans ce chapitre sont en grande partie préliminaires et indiquent essentiellement que le processus de découverte proposé alliant modèle

de soi du robot et approche par transférabilité est envisageable et potentiellement rapide. Afin de les valider de manière plus solide, deux aspects principaux sont à envisager :

- L'approche est-elle compétitive, en terme de qualité du contrôleur trouvé et de temps expérimental nécessaire, en comparaison des méthodes d'optimisation habituellement utilisées dans la littérature comme les techniques d'apprentissage par renforcement (e.g. méthodes par recherche locale de politique [Kohl & Stone 2004] ou basées sur du *Q-learning* [Minato & Asada 2000]).
- Comment peut-on détecter une nouvelle situation ? Une telle détection est facile quand le comportement du robot devient inefficace ou moins performant. Néanmoins, ce test peut ne pas être suffisant : en imaginant qu'un robot marche sur de l'herbe et repasse sur un sol plat, son comportement sera toujours aussi efficace. Dans un tel cas de figure, un autre moyen de détection doit être utilisé afin de relancer un processus de découverte. Il serait par exemple possible de tester régulièrement des comportements optimisés dans les environnements précédents pour comparer leur efficacité avec celle du comportement actuel. Si un "ancien" comportement se révèle être plus performant que le comportement actuel, il y a eu un changement de situation et le processus de découverte doit être relancé.

6.4.4 Conclusion

Dans ce chapitre, nous avons montré que le problème d'adaptation d'un robot à un nouvel environnement peut être reformulé comme un problème de passage à la réalité entre le modèle de soi du robot et le nouvel environnement. Un processus de découverte de contrôleurs utilisant l'approche par transférabilité a ensuite été proposé pour permettre au robot de s'adapter rapidement à de nouvelles situations. Une application sur un robot à locomotion roues-pattes a ensuite permis de valider cette approche dans un dispositif expérimental simple : le processus de découverte permet au robot de s'adapter rapidement à trois situations impliquant la découverte de modes de locomotion très différents.

Chapitre 7

Discussion et perspectives

7.1 Contributions

Au début de ce manuscrit, nous nous sommes posés la question suivante : *comment obtenir des solutions performantes dans un environnement complet donné en les optimisant principalement dans un environnement simplifié ?* Pour y répondre, nous avons considéré trois cas d'étude tirés de la robotique évolutionniste où, dans le cadre d'un processus d'optimisation de contrôleurs, la question du passage d'un environnement simplifié à un environnement complet est cruciale.

Les problèmes du passage à la réalité ou *reality gap* peuvent intervenir dans toute application robotique où les contrôleurs sont évalués en simulation plutôt que sur le robot. Si la simulation ne modélise pas parfaitement le robot, l'environnement du robot et/ou le comportement de ses capteurs, une solution performante en simulation peut être totalement inefficace en réalité. Plus précisément, la correspondance entre le comportement simulé et le comportement observé sur le robot n'est pas systématique, ce qui se traduit dans de nombreux travaux par une chute de performance lorsque la solution finale est transférée sur le robot. Dans ce contexte, nous proposons d'optimiser les contrôleurs en simulation via une approche par transférabilité : chaque contrôleur est à la fois évalué par une ou plusieurs valeurs de performance et une valeur de transférabilité qui indique s'il va se transférer plus ou moins bien sur le robot, le but étant de trouver des contrôleurs à la fois performants en simulation et transférables sur le robot. La fonction de transférabilité est apprise via la construction d'un modèle de substitution à partir de quelques expériences de transfert effectuées sur le robot au cours de l'optimisation. Ce modèle permet d'estimer une valeur approchée de transférabilité pour chacun des contrôleurs optimisés à partir d'une description de son comportement en simulation. Deux dispositifs expérimentaux robotiques nous ont permis de valider l'approche par transférabilité. Le premier consistait en la navigation d'un robot mobile e-puck dans un labyrinthe et le second en l'optimisation de comportements de marche pour un robot quadrupède. Dans les deux cas, notre approche a permis de trouver des contrôleurs performants sur le robot tout en ne nécessitant qu'une dizaine d'expériences en réalité au cours de l'optimisation pour apprendre la fonction de transférabilité. L'approche s'est montrée également compétitive comparée à plusieurs méthodes de l'état de l'art : des optimisations en simulation, différents types d'optimisations effectuées directement sur le robot et une approche basée bruit inspirée de la méthodologie des simulations minimales de Jakobi [Jakobi 1998a].

Le deuxième point que nous avons traité concerne la possibilité de doter des

contrôleurs de capacités de généralisation pendant un processus d'optimisation, c'est-à-dire d'obtenir des contrôleurs par optimisation qui soient robustes aux spécifités du contexte d'évaluation utilisé. En nous inspirant d'une méthodologie classique en apprentissage supervisé [Alpaydin 2004], nous avons proposé qu'un contrôleur soit estimé sur trois ensembles de contextes : pendant l'optimisation sur un contexte d'apprentissage et un ensemble de contextes de validation ; après l'optimisation, sur un ensemble de contextes de test pour estimer ses capacités de généralisation effectives. Le contexte d'apprentissage fait office d'environnement simplifié et les ensembles de validation et de test d'environnement complet. Afin d'obtenir des contrôleurs dotés de capacités de généralisation en un temps d'évaluation limité, nous avons ensuite présenté l'approche *ProGAb* (*Promoting the Generalisation Ability*) basée sur l'approche par transférabilité : chaque contrôleur est évalué sur le contexte d'apprentissage et sa performance sur l'ensemble de validation est apprise via un modèle de substitution construit pendant l'optimisation à partir de l'évaluation de quelques contrôleurs sur l'ensemble de validation. Ce modèle fournit une approximation de la performance des contrôleurs sur l'ensemble de validation à partir de leurs comportements sur l'ensemble d'apprentissage. L'approche *ProGAb* a été validée sur deux dispositifs expérimentaux robotiques en simulation. Le premier est inspiré d'une expérience de neurosciences sur la mémoire de travail et consiste en la navigation d'un agent mobile dans un labyrinthe en T. Le second simule une tâche complexe de collecte de balles dans une arène. Dans les deux cas, l'approche *ProGAb* a été comparée à plusieurs méthodes plus classiques, comme optimiser uniquement sur le contexte d'apprentissage, optimiser sur l'ensemble de validation, ou encore optimiser sur un nombre fixe de contextes éventuellement variables d'une génération à l'autre. Il s'avère que notre approche permet de trouver des contrôleurs dotés des meilleures capacités de généralisation, et ce en un nombre d'évaluations limité comparé à ces autres méthodes.

Enfin, dans des travaux plus préliminaires, nous nous sommes intéressés au problème de l'adaptation d'un robot mobile à de nouvelles situations, par exemple à un changement d'environnement. Quand le comportement du robot devient inefficace, il lui est possible de lancer un processus de découverte, via une optimisation par exemple, pour explorer le nouvel espace de performance et trouver ainsi un contrôleur efficace dans la nouvelle situation. Après avoir remarqué l'intérêt d'assurer ce processus de découverte en s'aidant d'un modèle de soi du robot, notamment en terme de coût expérimental et de rapidité du mécanisme d'adaptation, nous nous sommes interrogés sur la nécessité de reconstruire un modèle à chaque nouvelle situation comme c'est le cas dans les travaux actuels sur le sujet [Bongard *et al.* 2006]. Dans le cas où le modèle de soi du robot reste suffisamment fiable dans la nouvelle situation, nous proposons de reformuler le problème d'adaptation du robot comme un problème de transférabilité entre le modèle de soi et le robot dans la nouvelle situation. Cette approche a notamment été considérée dans le cadre de la locomotion hybride roues-pattes où la découverte d'un mode de locomotion adapté dans un nouvel environnement reste une question ouverte. Le dispositif expérimental consistait en l'optimisation de contrôleurs de marche pour un robot quadrupède hybride,

avec des résultats prometteurs où le robot s'adapte à de nouvelles contraintes environnementales en quelques minutes.

Dans ces trois cas d'étude, l'approche mise en place est basée sur les points principaux suivants :

- L'optimisation se fait majoritairement dans l'environnement simplifié (simulation, contexte d'apprentissage, modèle de soi), d'où un processus rapide et accélérable.
- On ne cherche pas à améliorer l'environnement simplifié pour le faire coller à l'environnement complet : on exploite simplement l'environnement simplifié là où il semble fiable. Autrement dit, on n'apprend pas un modèle, mais un critère de correspondance comportementale entre environnement simplifié et environnement complet.
- Pour ce faire, la performance dans l'environnement complet est apprise au cours de l'optimisation par un modèle de substitution à partir de quelques expériences effectuées dans l'environnement complet. Le coût expérimental d'une telle approche est donc limité, notamment par rapport aux approches où les solutions sont évaluées directement dans l'environnement complet pour éviter de s'appuyer sur un environnement simplifié trompeur [Brooks 1992].
- Le modèle de substitution prend en entrée une description comportementale des contrôleurs dans l'environnement simplifié.
- L'utilisation d'algorithmes d'optimisation multi-objectifs nous permet de gérer plusieurs objectifs de performance potentiellement conflictuels.

Du point de vue de la robotique, nous avons proposé une méthode automatique d'optimisation de contrôleurs simple et rapide qui prend explicitement en compte les problèmes de passage de la simulation à la réalité. Les différentes contributions de ce travail de thèse s'axent suivant les 5 points suivants :

- L'approche par transférabilité permet de concevoir des algorithmes d'optimisation automatique rapides sans avoir à expliciter les causes des écarts observés entre simulation et réalité, ni chercher à faire coller le simulateur à la réalité.
- Une telle approche est directement applicable à l'adaptation d'un robot à son environnement basée sur un modèle de soi et les résultats dans ce contexte nous semblent très prometteurs en terme de rapidité et de performance du mécanisme d'adaptation.
- Suivant une approche similaire, nous avons montré la possibilité d'obtenir par optimisation des contrôleurs dotés de capacités de généralisation avec un coût computationnel relativement faible.
- L'ensemble de nos résultats souligne les avantages apportés par l'utilisation d'un modèle de substitution dans un cadre robotique, et ce malgré l'utilisation de techniques d'interpolation et de stratégies de construction simples.
- Plusieurs comparaisons dans les chapitres 4 et 5 indiquent l'intérêt de s'appuyer sur une description comportementale des solutions plutôt que génotypiques pour construire un modèle de substitution. Ce point sera discuté plus en détail dans la partie 7.4.

Hors de ces contributions spécifiques liées au domaine de la robotique, l'approche

par transférabilité propose également une méthodologie permettant de réconcilier simulateurs (i.e. d'environnements simplifiés) et processus d'optimisation d'une manière plus générale via la définition d'une fonction de transférabilité. Nous pensons en effet que cette approche par transférabilité peut être intégrée à tout processus d'optimisation où l'évaluation des solutions se fait dans un environnement simplifié, et ce afin d'obtenir notamment un processus d'optimisation rapide pouvant découvrir des solutions originales au problème considéré. Une contrainte importante de cette approche est qu'elle nécessiterait de devoir effectuer quelques expériences dans l'environnement complet pendant l'optimisation, ce qui implique d'avoir accès à l'environnement complet en question au cours du processus.

Suite à ce bref rappel des différents résultats et des contributions de ce travail de thèse, nous allons maintenant discuter trois perspectives importantes :

- Comment compléter le mécanisme d'adaptation présenté dans le chapitre 6 dans le cas d'un robot mobile ?
- Comment améliorer l'environnement simplifié à partir de la fonction de transférabilité ?
- Comment l'idée de construire un modèle de substitution sur un espace comportemental peut-elle aider à étudier des systèmes complexes ?

7.2 Vers un mécanisme d'adaptation complet

Dans le chapitre précédent, nous avons présenté un mécanisme d'adaptation d'un robot possédant un modèle de soi, l'idée étant d'appliquer une approche par transférabilité pour trouver des contrôleurs performants d'après le modèle de soi du robot et transférables du modèle au robot dans chaque nouvelle situation. Cette approche n'est a priori applicable que si le modèle de soi du robot est suffisamment fiable dans la situation rencontrée par le robot.

Afin de compléter ce mécanisme, on peut distinguer trois types de phénomènes auxquels le robot sera confronté :

- (1) des événements ponctuels ;
- (2) une situation où le modèle de soi du robot est relativement fiable : par exemple, des variations environnementales modérées (passage d'un terrain plat à un terrain herbeux) ou des modifications morphologiques graduelles (fatigue des moteurs, déformations mécaniques, perte d'une patte avec un robot suffisamment redondant pour conserver une dynamique proche)
- (3) des variations environnementales drastiques (environnement complètement différent) ou des modifications morphologiques importantes à moyen ou long terme (perte d'une patte, panne d'un moteur).

Dans le cas 1, on ne se situe pas vraiment dans un cas d'adaptation. Par exemple, en supposant que le robot est confronté à un obstacle ponctuel qui n'est pas prévu dans son modèle, il sera probablement préférable d'utiliser une stratégie pré-définie pour franchir l'obstacle : dans ce cas, le processus est très rapide vu qu'il n'y a pas d'apprentissage à faire. Il est également envisageable d'utiliser les capteurs du robot

(par exemple une Kinect) pour détecter l'obstacle et l'intégrer dans la simulation. Le robot peut alors développer, en ligne, une stratégie d'évitement. Le point important dans ce premier cas est que le robot n'a pas à adapter son comportement.

Si le robot change de situation (cas 2 ou 3), son modèle de soi peut rester plus ou moins fiable suivant les changements intervenus. Il est alors possible d'employer le processus suivant :

- (A) Tentative d'adaptation via l'approche par transférabilité. Si le modèle de soi du robot est suffisamment fiable (cas 2), le robot devrait découvrir des comportements adaptés à la nouvelle situation.
- (B) Si cette première tentative échoue et qu'aucun comportement efficace sur le robot n'a été trouvé, cela peut signifier que le modèle de soi du robot n'est pas suffisamment fiable (cas 3) et qu'il lui est nécessaire de ré-apprendre un nouveau modèle ou d'améliorer le modèle existant.

Même si le mécanisme d'adaptation A échoue, il permet de générer des données sur le robot dans sa situation actuelle qui peuvent être ré-utilisées ensuite comme base pour apprendre un nouveau modèle de soi (mécanisme B). Ce second mécanisme interviendrait par exemple quand le robot change complètement d'environnement. On peut néanmoins remarquer qu'un robot mobile donné est construit pour se déplacer dans un certain nombre d'environnements. Par exemple, un robot amphibie est conçu pour pouvoir se mouvoir sur terre et dans l'eau. Il est donc envisageable que le concepteur du robot fournisse un modèle de soi au robot par type d'environnement qu'il sera sensé explorer (terrestre, aquatique, aérien, terrain volcanique, vide spatial, ...). Dans un tel cas de figure, avant d'utiliser le mécanisme A, le robot devra déterminer via ses capteurs dans quel type d'environnement il se trouve et donc quel modèle de soi utiliser. Si le robot se retrouve dans un environnement où aucun modèle de soi correct n'est connu ou s'il subit des modifications morphologiques importantes, le mécanisme A devrait échouer et il sera alors nécessaire d'apprendre un nouveau modèle via le mécanisme B. L'apprentissage du nouveau modèle pourra par exemple se faire à partir de la fonction de transférabilité approchée obtenue, ce que nous discuterons dans la partie 7.3.

Le processus proposé a l'avantage de fournir deux mécanismes d'adaptation : (1) un mécanisme rapide pour de faibles variations environnementales et morphologiques via l'approche par transférabilité sans que le modèle de soi ne soit modifié ; (2) un mécanisme plus lent d'apprentissage d'un nouveau modèle de soi en cas d'environnement inconnu et non prévu ou si la morphologie du robot est sérieusement modifiée. Un tel processus est assez proche du mécanisme proposé par Zagal et Lipson dans l'article [Zagal & Lipson 2011] où le robot est doté de deux contrôleurs : un contrôleur "inné" rapide qui contient tous les comportements possibles du robot et un contrôleur "songeur" plus lent qui va déterminer quelles parties du contrôleur "inné" utiliser en fonction de l'environnement. Dans notre travail, au lieu de manipuler des contrôleurs, nous manipulons directement le modèle de soi du robot. Nous ne sommes donc pas limités à un ensemble de comportements pré-déterminés contenu dans un contrôleur "inné", mais à tous les comportements modélisés par le modèle de soi.

Le processus proposé ici introduit des mécanismes d'adaptation sur des échelles de temps différentes, un phénomène qui a déjà été observé en biologie. Par exemple, d'après [Kording *et al.* 2007], le mécanisme d'adaptation à la fatigue d'un muscle est plus rapide que le mécanisme d'adaptation intervenant quand un muscle se développe ou subit des dommages. Les auteurs s'appuient notamment sur des données d'adaptation de saccades oculaires chez le singe. Il a par exemple été observé qu'en cas de variations rapides mais faibles le singe est capable de s'adapter d'un essai à l'autre au cours d'une même séquence de tests, alors qu'en cas de variations plus lentes et importantes, un apprentissage lent se met en place d'un jour à l'autre. A première vue, ces observations sont plutôt en accord avec le processus proposé : on cherche d'abord rapidement un comportement à peu près efficace à partir de la connaissance acquise sur la tâche (i.e. un modèle de la tâche), puis de manière plus lente, on peut mettre à jour cette connaissance pour capter des variations importantes et durables.

Un tel découpage en deux mécanismes d'adaptation distincts n'est pas non plus sans analogie avec l'apprentissage de certains comportements chez les animaux. Par exemple, un grand nombre de mammifères quadrupèdes ont la capacité de nager de manière presque innée (l'apprentissage est quasi-instantané) et le comportement qu'ils utilisent est proche d'un comportement de marche. Dans ce cas, on pourrait imaginer que l'animal découvre rapidement, via un modèle de sa locomotion, des comportements de "marche" terrestre qui se transfèrent bien dans un environnement aquatique, sans que la construction d'un modèle de locomotion valide dans l'eau soit nécessaire. A l'opposé, les hominidés (grands singes et humains) ne sont pas capables de nager de manière innée de par leurs morphologies : leurs locomotions terrestres sont peu efficaces dans un milieu aquatiques. Une phase d'apprentissage plus longue leur est nécessaire pour pouvoir apprendre à nager correctement. Ceci pourrait alors correspondre à la nécessité de construire un autre modèle de soi qui soit valide dans l'environnement aquatique.

Ces différentes ressemblances avec des phénomènes d'adaptation biologiques nous confortent dans l'idée que le processus proposé ici pourrait être tout à fait efficace sur un robot en terme de rapidité et de versatilité : le robot pourrait s'adapter rapidement quand son modèle de soi est suffisant ; dans le cas contraire, le robot pourrait s'adapter également en construisant un nouveau modèle via un mécanisme plus lent. Il sera bien évidemment nécessaire d'effectuer de nombreuses réflexions et expériences supplémentaires pour valider un tel processus.

7.3 Quand la simulation apprend de la transférabilité

Afin de s'assurer que des solutions optimisées dans l'environnement simplifié soient effectivement efficaces dans l'environnement complet, certains travaux proposent d'améliorer le modèle $\mathcal{M}$ utilisé comme environnement simplifié [Bongard *et al.* 2006, Zagal *et al.* 2008]. En améliorant ou en ré-apprenant le modèle simplifié, on aura tendance à renforcer la correspondance entre les com-

portements simulés et ceux observés en réalité. Afin d'éviter un apprentissage de modèle qui peut s'avérer long et coûteux en temps expérimental, l'approche par transférabilité proposée ici repose sur l'optimisation des solutions directement dans le modèle simplifié fixe et cherche, via la construction d'une fonction de transférabilité, à exploiter les parties de ce modèle $\mathcal{M}$ qui correspondent à des comportements proches de ceux observés dans l'environnement complet.

Une fois qu'une approximation de la fonction de transférabilité a été construite, il est facile de déterminer les comportements qui vont correspondre à de faibles valeurs de transférabilité et donc de localiser les parties de l'espace comportemental dans l'environnement simplifié où les écarts prédits avec la réalité sont importants. Néanmoins, il est plus difficile de faire le lien entre ces zones non transférables et les phénomènes physiques que le simulateur simule mal ou ne simule pas (voir [Ulam *et al.* 2005]).

Pour trouver un meilleur modèle, il serait par exemple possible de lancer une optimisation en se servant des données générées sur le système à modéliser pendant la construction de la fonction de transférabilité. Mais cette fonction nous donne de l'information supplémentaire : elle permet de distinguer les comportements bien simulés par le modèle $\mathcal{M}$ des comportements mal simulés par $\mathcal{M}$. Ainsi, un modèle $\mathcal{M}'$ qui modélise mieux le système considéré que $\mathcal{M}$ doit vérifier les 2 critères suivants :

- (C1) modéliser de la même manière les parties transférables de $\mathcal{M}$;
- (C2) modéliser de manière différente les parties non-transférables de $\mathcal{M}$, mais en accord avec les données générées sur le système.

Pour évaluer ces aspects, il est possible d'effectuer un échantillonnage de l'espace d'entrée des deux modèles et de calculer pour chacun des points i une valeur de distance d_i entre les comportements observés avec $\mathcal{M}$ et $\mathcal{M}'$. On peut alors définir 3 objectifs pour évaluer $\mathcal{M}'$:

- (O1) minimiser la somme des d_i sur les i correspondant à des comportements transférables dans $\mathcal{M}$;
- (O2) maximiser la somme des d_i sur les i correspondant à des comportements non-transférables dans $\mathcal{M}$;
- (O3) minimiser l'erreur de prédiction sur les données réelles correspondant à des comportements non-transférables dans $\mathcal{M}$.

Un tel processus permet de s'assurer que le modèle $\mathcal{M}'$ récupère à la fois les informations des parties intéressantes de $\mathcal{M}$ (objectif O1), explore de nouvelles façons de simuler le système dans les zones non fiables de $\mathcal{M}$ (objectif O2) avec la contrainte de coller aux données disponibles dans ces même zones (objectif O3). Dans un tel cadre, l'approche par transférabilité pourrait être une méthode originale pour valider des modèles via la construction rapide d'une fonction de transférabilité à partir de quelques expériences sur le système modélisé et son utilisation pour extraire les parties pertinentes d'un modèle existant.

7.4 Améliorer la compréhension des systèmes complexes

L'approche par transférabilité construit un modèle de substitution de la fonction de transférabilité exacte au cours de l'optimisation (ou de la capacité de généralisation dans l'approche *ProGAb*). Nous avons choisi d'utiliser comme espace d'entrée de ce modèle une description comportementale des contrôleurs dans l'environnement simplifié au lieu d'une description génotypique plus habituelle dans le domaine des algorithmes évolutionnistes. Ce choix se basait notamment sur plusieurs travaux prouvant, dans des expériences de robotique évolutionniste, l'intérêt de comparer des comportements plutôt que des génotypes (e.g. [Mouret & Doncieux 2011]). La qualité des résultats obtenus sur les trois problèmes considérés — passage à la réalité, généralisation et adaptation —, ainsi que plusieurs comparaisons avec des méthodes où le modèle de substitution utilise le génotype en entrée justifient ce choix *a posteriori*.

En construisant le modèle de substitution sur un espace comportemental, on fait l'hypothèse que le comportement des solutions dans l'environnement simplifié permet de prédire ce qu'il se passe dans l'environnement complet. Cette hypothèse est loin d'être triviale. Par exemple dans le cas de la généralisation, cela signifie que le comportement dans un seul contexte d'apprentissage doit permettre de déterminer la capacité de généralisation sur les contextes de validation (pour une discussion détaillée sur ce point spécifique, voir partie 5.7.1). Les résultats indiquent que cette hypothèse est vérifiée dans les expériences présentées dans ce manuscrit.

Comme nous l'avons vu précédemment, l'approche par transférabilité peut être utilisée de deux façons : pour optimiser des solutions dans un environnement d'évaluation simplifié ou dans le cadre d'un processus de modélisation comme discuté dans la partie précédente. De par sa conception, elle est également idéale pour l'étude de systèmes complexes qui impliquent des temps expérimentaux importants. Enfin, il s'agit d'une méthode interactive dans le sens où l'utilisateur apporte de l'information sur le système étudié quand il définit la description comportementale à utiliser pour construire le modèle de substitution de la transférabilité. Il peut intégrer directement son expertise du système complet en décidant sur quelles bases il faut différencier les comportements du système et ainsi biaiser le processus d'optimisation ou de modélisation vers un type de comportement particulier.

Pour ces différentes raisons, l'approche par transférabilité n'est pas seulement une méthode d'optimisation rapide utilisable dans des applications robotiques, mais apparaît également comme une méthodologie simple et flexible pour l'étude et la modélisation de systèmes complexes en général.

7.5 Conclusions

L'approche par transférabilité présentée dans ce manuscrit de thèse apporte une réponse unifiée à différents problèmes de robotique évolutionniste. Au-delà de ces contributions spécifiques, nous pensons qu'elle peut s'appliquer à tout problème d'optimisation où l'évaluation des solutions se fait dans un environnement simplifié.

A partir des résultats obtenus, diverses extensions de ce travail ont été discutées, que ce soit pour concevoir de nouveaux processus d'adaptation en robotique mobile, pour construire ou améliorer des modèles de simulation ou encore pour mieux comprendre le fonctionnement de systèmes complexes. Même si nos futurs travaux vont plutôt s'attacher à développer les aspects robotiques (notamment concernant l'adaptation), nous espérons également pouvoir valider ces différentes intuitions dans d'autres domaines de recherche.

Chapitre 8

Conclusions

En robotique, le contrôleur d'un robot est souvent obtenu par des méthodes de théorie du contrôle à partir d'un système d'équations différentielles décrivant sa dynamique. Il est néanmoins difficile pour le concepteur de déterminer ce que le robot perçoit de son environnement, ainsi que les phénomènes sensoriels et dynamiques spécifiques qu'il peut exploiter [Brooks 1992]. Partant de cette remarque, plusieurs domaines de recherche, dont la robotique évolutionniste, se sont intéressés à développer des méthodes automatiques de conception de contrôleurs robotiques sans intervention du concepteur. La performance d'un contrôleur est alors directement estimée par une expérience sur le robot ou dans un simulateur ; on peut alors par exemple chercher les contrôleurs qui maximisent cette valeur de performance.

La localisation de l'étape d'évaluation pour un tel processus d'optimisation est sujette à débat : faut-il estimer la performance des contrôleurs sur le robot ou en simulation ? On peut résumer les arguments pour l'un ou pour l'autre de la manière suivante :

- Des expériences sur un robot sont limitées au temps réel. Or, un processus d'optimisation peut nécessiter de nombreuses expériences et effectuer les différentes évaluations sur le robot peut donc être très long. D'autre part, un processus d'optimisation correct pourra difficilement être obtenu avec des contraintes drastiques sur le nombre d'évaluations.
- Des expériences en simulation peuvent être rapides, parallélisables et accélérées. Néanmoins, un simulateur ne modélise jamais parfaitement les comportements du robot. Optimiser en simulation peut ainsi mener à des contrôleurs apparemment efficaces, mais qui se révèlent sans intérêt quand ils sont testés sur le robot : c'est le problème du passage à la réalité ou *reality gap*.

Il y a donc un antagonisme entre le temps d'évaluation disponible (i.e. la durée du processus d'optimisation) et la qualité de l'évaluation (i.e. la capacité à prédire la performance réelle). Ce problème n'est pas spécifique à la robotique, car il concerne potentiellement tout processus d'optimisation ou d'apprentissage où évaluer dans l'environnement complet entraîne des expériences trop longues, d'où une évaluation des solutions dans un environnement simplifié.

Cette thèse cherche à répondre à la problématique générale suivante : comment peut-on optimiser des solutions dans un environnement simplifié qui vont être performantes dans l'environnement complet ciblé ? Pour répondre à cette question, nous avons introduit l'approche par transférabilité qui repose sur les principes suivants :

- Les solutions sont optimisées essentiellement dans l'environnement simplifié.
- Une fonction de transférabilité permet de déterminer si un individu dans l'en-

vironnement simplifié va effectivement bien se transférer dans l'environnement complet.

- Cette fonction est approchée via la construction d'un modèle de substitution à partir de quelques expériences effectuées dans l'environnement complet pendant l'optimisation. Ce modèle de substitution est utilisé pour prédire la valeur de transférabilité associée à chaque solution non testée dans l'environnement complet en fonction de son comportement dans l'environnement simplifié.
- Chaque solution est alors évaluée par un ou plusieurs objectifs de performance dans l'environnement simplifié et une valeur de transférabilité approchée dans le cadre d'un processus d'optimisation multi-objectif. Le but est alors de trouver des individus performants dans l'environnement simplifié et transférables de l'environnement simplifié à l'environnement complet.

La validité d'une telle approche a été considérée dans trois cas d'étude inspirés de la robotique évolutionniste où le passage d'un environnement simplifié à un environnement complet s'avère crucial. Chaque cas est développé dans un chapitre de ce manuscrit.

Dans le chapitre 4, nous avons tout d'abord considéré les problèmes de passage de la simulation à la réalité : comment en optimisant des contrôleurs en simulation peut-on obtenir des contrôleurs performants sur le robot ? Sur deux dispositifs expérimentaux robotiques — l'un considérant la navigation d'un robot mobile e-puck dans un labyrinthe et l'autre l'optimisation de contrôleurs de marche pour un robot quadrupède — l'approche par transférabilité a permis de trouver des contrôleurs à la fois efficaces et transférables sur le robot. De plus, le processus d'optimisation ne nécessite qu'une dizaine d'expériences en réalité, soit un coût restreint en temps expérimental. Ces premiers résultats prouvent l'intérêt de l'approche comme méthode d'optimisation rapide applicable à des problèmes robotiques, notamment en comparaison de méthodes plus classiques tirées de l'état de l'art : optimiser directement sur le robot, optimiser uniquement en simulation et utiliser une approche basée bruit inspirée de la méthodologie des simulations minimales de Jakobi [Jakobi 1998a].

Dans le chapitre 5, nous avons ensuite considéré la question de comment doter des contrôleurs obtenus par optimisation de capacités de généralisation. Nous nous sommes tout d'abord inspiré de la méthodologie utilisée en apprentissage supervisé pour évaluer les capacités de généralisation dans un cadre général. Nous avons ensuite proposé d'optimiser les contrôleurs sur un petit ensemble de contextes d'apprentissage (environnement simplifié) et qu'un grand ensemble de contextes de validation fasse office d'environnement complet. Ici, le critère de transférabilité correspond à la capacité de généralisation du contrôleur considéré sur l'ensemble de validation. A la fin d'une optimisation, la capacité de généralisation des meilleures solutions obtenues est estimée sur un troisième ensemble disjoint des deux autres, l'ensemble de contextes de test. Deux dispositifs expérimentaux robotiques en simulation ont été considérés : une tâche de navigation d'un robot mobile dans un labyrinthe inspirée d'une expérience de neurosciences ; un tâche complexe de collecte de balles dans une arène. Comparée à plusieurs méthodes de l'état de l'art, l'approche proposée permet de trouver des capacités de généralisation significative-

ment meilleures tout en assurant un nombre d'évaluations raisonnable.

Enfin, dans des travaux plus préliminaires présentés dans le chapitre 6, nous avons considéré le problème de l'adaptation du comportement d'un robot à une nouvelle situation, par exemple à un nouvel environnement. Nous avons tout d'abord souligné l'intérêt de gérer ce mécanisme d'adaptation par optimisation à l'aide d'un modèle de soi du robot afin de découvrir de nouveaux comportements efficaces dans la nouvelle situation [Bongard *et al.* 2006]. Nous avons alors pu reformuler le processus d'adaptation comme un problème de transférabilité entre le modèle du soi du robot d'une part et le robot dans la nouvelle situation d'autre part. Considérant le cas d'un robot quadrupède à locomotion hybride roues-pattes devant s'adapter à des changements environnementaux, l'utilisation de l'approche par transférabilité sur un modèle de soi du robot a permis au robot de s'adapter en moins de deux minutes à chaque situation rencontrée. Dans ce chapitre, l'intérêt des résultats est double : nous avons proposé une solution partielle au problème de la locomotion hybride roues-pattes, une question ouverte en robotique ; nous avons aussi montré que l'approche par transférabilité pouvait être appliquée dans un contexte d'adaptation robotique en ligne, notamment de par sa rapidité.

Ces trois contributions dans le domaine robotique soulignent à la fois l'efficacité et la polyvalence de l'approche par transférabilité qui apparaît comme une méthode simple et rapide pour réconcilier simulateurs et optimisation.

Bibliographie

[Abbeel *et al.* 2006] P. Abbeel, V. Ganapathi and A. Ng. *Learning vehicular dynamics, with application to modeling helicopters.* Advances in Neural Information Processing Systems, vol. 18, page 1, 2006. (Cité en page 162.)

[Abbeel *et al.* 2007] P. Abbeel, A. Coates, M. Quigley and A. Ng. *An application of reinforcement learning to aerobatic helicopter flight.* In Proceedings of NIPS, page 1. MIT Press, 2007. (Cité en pages 40, 41, 42 et 162.)

[Abbeel *et al.* 2009] P. Abbeel, A. Coates, T. Hunter and A. Ng. *Autonomous autorotation of an RC helicopter.* In Experimental Robotics, pages 385–394. Springer, 2009. (Cité en pages 40 et 42.)

[Adigbli *et al.* 2007] P. Adigbli, C. Grand, J.-B. Mouret and S. Doncieux. *Nonlinear Attitude and Position Control of a Micro Quadrotor using Sliding Mode and Backstepping Techniques.* In 7th European Micro Air Vehicle Conference (MAV07), Toulouse, France, 2007. (Cité en pages 1 et 74.)

[Albu-Schaffer & Hirzinger 2001] A. Albu-Schaffer and G. Hirzinger. *Parameter identification and passivity based joint control for a 7 DOF torque controlled light weight robot.* In Proceedings of ICRA 2001, volume 3, pages 2852–2858. IEEE, 2001. (Cité en page 162.)

[Alliot 1996] J.M. Alliot. *Techniques d'optimisation stochastique appliquées aux problèmes du contrôle aérien.* HDR thesis, UniversitÃ© de Toulouse Paul Sabatier, 1996. (Cité en page 62.)

[Alpaydin 2004] E. Alpaydin. Introduction to machine learning. The MIT Press, 2004. (Cité en pages 43, 130, 131 et 176.)

[Azimi-Zonooz *et al.* 1989] A. Azimi-Zonooz, WF Krajewski, DS Bowles and DJ Seo. *Spatial rainfall estimation by linear and non-linear co-kriging of radar-rainfall and raingage data.* Stochastic Hydrology and Hydraulics, vol. 3, no. 1, pages 51–67, 1989. (Cité en page 59.)

[Bäck & Hoffmeister 1991] T. Bäck and F. Hoffmeister. *Extended Selection Mechanisms in Genetic Algorithms.* In Proceedings of the Fourth International Conference on Genetic Algorithms, pages 92–99. Morgan Kaufmann, 1991. (Cité en pages 13 et 32.)

[Bäck & Schwefel 1993] T. Bäck and H.P. Schwefel. *An overview of evolutionary algorithms for parameter optimization.* Evolutionary computation, vol. 1, no. 1, pages 1–23, 1993. (Cité en pages 7, 9 et 10.)

[Bäck *et al.* 1991] T. Bäck, F. Hoffmeister and H.P. Schwefel. *A survey of evolution strategies.* Proceedings of the 4th International Conference on Genetic ALgorithms and their Applications, 1991. (Cité en pages 7 et 12.)

[Back 1994] T. Back. *Selective pressure in evolutionary algorithms : A characterization of selection mechanisms.* In Proceedings of IEEE WCCI, pages 57–62. IEEE, 1994. (Cité en pages 12 et 13.)

[Bäck 1996] T. Bäck. Evolutionary algorithms in theory and practice. Oxford University Press New York, 1996. (Cité en page 7.)

[Bader & Zitzler 2011] J. Bader and E. Zitzler. *HypE : An algorithm for fast hypervolume-based many-objective optimization.* Evolutionary Computation, vol. 19, no. 1, pages 45–76, 2011. (Cité en page 26.)

[Baker 1985] J.E. Baker. *Adaptive selection methods for genetic algorithms.* In Proceedings of the 1st International Conference on Genetic Algorithms, pages 101–111. L. Erlbaum Associates Inc., 1985. (Cité en page 12.)

[Barate & Manzanera 2008] Renaud Barate and Antoine Manzanera. *Generalization performance of vision based controllers for mobile robots evolved with genetic programming.* Proceedings of GECCO, page 1331, 2008. (Cité en pages 45, 46, 129, 130 et 131.)

[Barshan & Durrant-Whyte 1995] B. Barshan and H.F. Durrant-Whyte. *Inertial navigation systems for mobile robots.* IEEE Transactions on Robotics and Automation, vol. 11, no. 3, pages 328–342, 1995. (Cité en page 171.)

[Barthelemy *et al.* 1994] J.F. Barthelemy*et al. Common misconceptions about neural networks as approximators.* Journal of computing in civil engineering, vol. 8, page 345, 1994. (Cité en page 53.)

[Barton 1998] R.R. Barton. *Simulation metamodels.* In Proceedings of the 30th conference on Winter simulation, pages 167–176. IEEE Computer Society Press, 1998. (Cité en page 51.)

[Bekele & Nicklow 2007] E.G. Bekele and J.W. Nicklow. *Multi-objective automatic calibration of SWAT using NSGA-II.* Journal of Hydrology, vol. 341, no. 3-4, pages 165–176, 2007. (Cité en page 28.)

[Berlanga *et al.* 2002] A Berlanga, A Sanchis, P Isasi and JM Molina. *Neural Network Controller against Environment : A Coevolutive approach to Generalize Robot Navigation Behavior.* Journal of Intelligent and Robotic Systems, pages 139–166, 2002. (Cité en pages 45, 46, 129 et 130.)

[Beume *et al.* 2007] N. Beume, B. Naujoks and M. Emmerich. *SMS-EMOA : Multiobjective selection based on dominated hypervolume.* European Journal of Operational Research, vol. 181, no. 3, pages 1653–1669, 2007. (Cité en page 26.)

[Birattari *et al.* 2002] M. Birattari, T. Stutzle, Luis Paquete and Klaus Varrentrapp. *A racing algorithm for configuring metaheuristics.* In Proceedings of GECCO, volume 2, pages 11–18, 2002. (Cité en page 47.)

[Bishop 1995] C.M. Bishop. Neural networks for pattern recognition. Oxford university press, 1995. (Cité en page 59.)

[Blickle & Thiele 1996] T. Blickle and L. Thiele. *A comparison of selection schemes used in evolutionary algorithms.* Evolutionary Computation, vol. 4, no. 4, pages 361–394, 1996. (Cité en pages 12 et 13.)

[Boeing & Bräunl 2007] A. Boeing and T. Bräunl. *Evaluation of real-time physics simulation systems.* In Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, pages 281–288. ACM New York, NY, USA, 2007. (Cité en page 100.)

[Bongard & Hornby 2010] J. Bongard and G. Hornby. *Guarding against premature convergence while accelerating evolutionary search.* Proceedings of GECCO, 2010. (Cité en page 47.)

[Bongard & Lipson 2004] J. Bongard and H. Lipson. *Once more unto the breach : Co-evolving a robot and its simulator.* In Proceedings of Artificial life IX, page 57. MIT Press, 2004. (Cité en page 40.)

[Bongard & Lipson 2005] J. Bongard and H. Lipson. *Nonlinear system identification using coevolution of models and tests.* IEEE Transactions on Evolutionary Computation, vol. 9, no. 4, pages 361–384, 2005. (Cité en pages 40, 49, 75, 162 et 171.)

[Bongard *et al.* 2006] J. Bongard, V. Zykov and H. Lipson. *Resilient machines through continuous self-modeling.* Science, vol. 314, no. 5802, page 1118, 2006. (Cité en pages 47, 48, 49, 75, 160, 161, 162, 171, 172, 176, 180 et 187.)

[Bongard 2007] J. Bongard. *Exploiting multiple robots to accelerate self-modeling.* In Proceedings of GECCO'07, pages 214–221. ACM, 2007. (Cité en page 3.)

[Bongard 2009] J. Bongard. *Accelerating self-modeling in cooperative robot teams.* Evolutionary Computation, IEEE Transactions on, vol. 13, no. 2, pages 321–332, 2009. (Cité en page 161.)

[Bongard 2011a] J. Bongard. *Innocent until proven guilty : Reducing robot shaping from polynomial to linear time.* Evolutionary Computation, IEEE Transactions on, no. 99, pages 1–15, 2011. (Cité en pages 31 et 47.)

[Bongard 2011b] J. Bongard. *Morphological change in machines accelerates the evolution of robust behavior.* PNAS, vol. 2010, January 2011. (Cité en page 45.)

[Borenstein *et al.* 1997] J. Borenstein, HR Everett, L. Feng and D. Wehe. *Mobile robot positioning : Sensors and techniques.* Journal of robotic systems, vol. 14, no. 4, pages 231–249, 1997. (Cité en page 171.)

[Braver *et al.* 1995] T.S. Braver, J.D. Cohen and D. Servan-Schreiber. *A computational model of prefrontal cortex function.* NIPS, pages 141–148, 1995. (Cité en page 138.)

[Brindle 1981] A. Brindle. *Genetic algorithms for function optimization.* PhD thesis, University of Alberta, Department of Computer Science, 1981. (Cité en page 12.)

[Brockhoff & Zitzler 2007] D. Brockhoff and E. Zitzler. *Improving hypervolume-based multiobjective evolutionary algorithms by using objective reduction methods.* In Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, pages 2086–2093. IEEE, 2007. (Cité en page 26.)

[Brooks 1991a] R.A. Brooks. *Intelligence without reason.* Artificial intelligence : critical concepts, vol. 3, 1991. (Cité en pages 1, 2 et 35.)

[Brooks 1991b] R.A. Brooks. *New approaches to robotics.* Science, vol. 253, no. 5025, page 1227, 1991. (Cité en pages 1, 2 et 35.)

[Brooks 1992] R.A. Brooks. *Artificial life and real robots.* In Proceedings of the first European conference on artificial life, pages 3–10, 1992. (Cité en pages 1, 2, 3, 29, 35, 73, 177 et 185.)

[Bubnicki 2005] Z. Bubnicki. Modern control theory. Springer-Verlag New York Inc, 2005. (Cité en page 1.)

[Buhmann 1993] M. Buhmann. *New developments in the theory of radial basis function interpolation.* PhD thesis, University of Cambridge, 1993. (Cité en page 58.)

[Buhmann 2000] M. Buhmann. *Radial basis functions.* Acta Numerica 2000, vol. 9, no. 1, pages 1–38, 2000. (Cité en pages 54 et 58.)

[Bunke & Shearer 1998] H. Bunke and K. Shearer. *A graph distance metric based on the maximal common subgraph.* Pattern recognition letters, vol. 19, no. 3-4, pages 255–259, 1998. (Cité en page 34.)

[Campbell *et al.* 2005] J. Campbell, R. Sukthankar, I. Nourbakhsh and A. Pahwa. *A robust visual odometry and precipice detection system using consumer-grade monocular vision.* In Proceedings of ICRA, pages 3421–3427. IEEE, 2005. (Cité en page 171.)

[Canudas-de-Wit *et al.* 1996] C. Canudas-de-Wit, B. Siciliano and G. Bastin. Theory of robot control. Springer Verlag, 1996. (Cité en page 1.)

[Charnes & Cooper 1957] A. Charnes and W.W. Cooper. *Management models and industrial applications of linear programming.* Management Science, vol. 4, no. 1, pages 38–91, 1957. (Cité en pages 16 et 19.)

[Chen *et al.* 2009] J. Chen, B. Xin, Z. Peng, L. Dou and J. Zhang. *Optimal Contraction Theorem for Exploration–Exploitation Tradeoff in Search and Optimization.* Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on, vol. 39, no. 3, pages 680–691, 2009. (Cité en page 13.)

[Christensen *et al.* 2010] D. Christensen, A. Spröwitz and A. Ijspeert. *Distributed online learning of central pattern generators in modular robots.* From Animals to Animats 11, pages 402–412, 2010. (Cité en pages 48, 49 et 160.)

[Cliff *et al.* 1993a] D. Cliff, P. Husbands and I. Harvey. *Evolving visually guided robots.* From animals to animats, vol. 2, pages 374–383, 1993. (Cité en page 37.)

[Cliff *et al.* 1993b] D. Cliff, P. Husbands and I. Harvey. *Explorations in evolutionary robotics.* Adaptive Behavior, vol. 2, no. 1, page 73, 1993. (Cité en pages 1 et 29.)

[Coello & Lamont 2004] C.A.C. Coello and G.B. Lamont. Applications of multi-objective evolutionary algorithms. World Scientific Pub Co Inc, 2004. (Cité en page 28.)

[Coello Coello 2006] C.A. Coello Coello. *Evolutionary multi-objective optimization : a historical view of the field.* Computational Intelligence Magazine, IEEE, vol. 1, no. 1, pages 28–36, 2006. (Cité en page 28.)

[Coello *et al.* 2007] C.A.C. Coello, G.B. Lamont and D.A. Van Veldhuizen. Evolutionary algorithms for solving multi-objective problems, volume 5. Springer-Verlag New York Inc, 2007. (Cité en pages 21 et 29.)

[Coello 1999] C.A.C. Coello. *A comprehensive survey of evolutionary-based multiobjective optimization techniques.* Knowledge and Information systems, vol. 1, no. 3, pages 129–156, 1999. (Cité en page 16.)

[Coello 2000a] C.A.C. Coello. *Constraint-handling using an evolutionary multiobjective optimization technique.* Civil engineering and environmental systems, vol. 17, pages 319–346, 2000. (Cité en page 29.)

[Coello 2000b] C.A.C. Coello. *Use of a self-adaptive penalty approach for engineering optimization problems.* Computers in Industry, vol. 41, no. 2, pages 113–127, 2000. (Cité en page 28.)

[Cressie 1990] N. Cressie. *The origins of kriging.* Mathematical Geology, vol. 22, no. 3, pages 239–252, 1990. (Cité en page 59.)

[Darwin 1859] C. Darwin. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*, 1859. (Cité en page 7.)

[Davidor 1991] Y. Davidor. Genetic algorithms and robotics : A heuristic strategy for optimization, volume 1. World Scientific Pub Co Inc, 1991. (Cité en pages 29 et 30.)

[De Jong *et al.* 2001] E. De Jong, R. Watson and J.B. Pollack. *Reducing bloat and promoting diversity using multi-objective methods.* In Proceedings of GECCO'01, pages 11–18. ACM, 2001. (Cité en pages 13 et 14.)

[De Jong 1975] K.A. De Jong. *Analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Computer and Communication Sciences Department, 1975. (Cité en pages 13 et 14.)

[De Jong 2006] K.A. De Jong. Evolutionary computation : a unified approach. MIT Press, 2006. (Cité en page 7.)

[Deb & Goldberg 1989] K. Deb and D.E. Goldberg. *An investigation of niche and species formation in genetic function optimization.* In Proceedings of the third international conference on Genetic algorithms, pages 42–50, 1989. (Cité en page 32.)

[Deb *et al.* 2000] K. Deb, S. Agrawal, A. Pratap and T. Meyarivan. *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization : NSGA-II.* In Parallel Problem Solving from Nature PPSN VI, pages 849–858. Springer, 2000. (Cité en pages 21 et 23.)

[Deb *et al.* 2002a] K. Deb, A. Anand and D. Joshi. *A computationally efficient evolutionary algorithm for real-parameter optimization.* Evolutionary computation, vol. 10, no. 4, pages 371–395, 2002. (Cité en page 9.)

[Deb *et al.* 2002b] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. *A fast and elitist multiobjective genetic algorithm : NSGA-II.* Evolutionary Computation, IEEE Transactions on, vol. 6, no. 2, pages 182–197, 2002. (Cité en pages 4, 21, 23, 24, 36, 94, 106, 139, 144 et 168.)

[Deb *et al.* 2002c] K. Deb, L. Thiele, M. Laumanns and E. Zitzler. *Scalable multi-objective optimization test problems.* In wcci, pages 825–830. IEEE, 2002. (Cité en page 27.)

[Deb *et al.* 2005] K. Deb, M. Mohan and S. Mishra. *Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions.* Evolutionary Computation, vol. 13, no. 4, pages 501–525, 2005. (Cité en pages 21, 24 et 25.)

[Deb 2001a] K. Deb. Multi-objective optimization using evolutionary algorithms, volume 16. Wiley, 2001. (Cité en pages 3, 9, 10, 15, 23, 24 et 29.)

[Deb 2001b] K Deb. Multi-objective optimization using evolutionary algorithms. John Wiley and Sons, 2001. (Cité en page 137.)

[Di Paolo 2000] E.A. Di Paolo. *Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions.* In From Animals to Animats 6 : Proceedings of SAB, 2000. (Cité en page 45.)

[Dissanayake *et al.* 2001] G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte and M. Csorba. *A solution to the simultaneous localization and map building (SLAM) problem.* Robotics and Automation, IEEE Transactions on, vol. 17, no. 3, pages 229–241, 2001. (Cité en page 171.)

[Doncieux & Mouret 2010] S. Doncieux and J.-B. Mouret. *Behavioral diversity measures for Evolutionary Robotics.* In Proceedings of IEEE-CEC'10, pages 1303–1310, 2010. (Cité en pages 32, 34, 81, 129, 130, 134, 137, 138, 143, 146 et 171.)

[Droste *et al.* 2003] S. Droste, T. Jansen, K. Tinnefeld and I. Wegener. *A new framework for the valuation of algorithms for black-box optimization.* Foundations of Genetic Algorithms, vol. 7, pages 253–270, 2003. (Cité en page 30.)

[Droste *et al.* 2006] S. Droste, T. Jansen and I. Wegener. *Upper and lower bounds for randomized search heuristics in black-box optimization.* Theory of Computing Systems, vol. 39, no. 4, pages 525–544, 2006. (Cité en page 30.)

[Drucker *et al.* 1997] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola and V. Vapnik. *Support vector regression machines.* Advances in neural information processing systems, pages 155–161, 1997. (Cité en page 131.)

[Duckstein 1981] L. Duckstein. *Multiobjective optimization in structural design : The model choice problem.* Rapport technique, DTIC Document, 1981. (Cité en pages 16 et 19.)

[Dyn & Ron 1995] N. Dyn and A. Ron. *Radial basis function approximation : from gridded centres to scattered centres.* Proceedings of the London Mathematical Society, vol. 3, no. 1, page 76, 1995. (Cité en page 58.)

[Eiben & Schippers 1998] A.E. Eiben and CA Schippers. *On evolutionary exploration and exploitation.* Fundamenta Informaticae, vol. 35, no. 1, pages 35–50, 1998. (Cité en pages 10, 12 et 13.)

[Eiben *et al.* 1998] A.E. Eiben, I.G. Sprinkhuizen-Kuyper and B.A. Thijssen. *Competing crossovers in an adaptive GA framework.* In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, pages 787–792. IEEE, 1998. (Cité en page 10.)

[Endo & Hirose 2000] G. Endo and S. Hirose. *Study on roller-walker (multi-mode steering control and self-contained locomotion).* In Proceedings of IEEE-ICRA'00, volume 3, pages 2808–2814. IEEE, 2000. (Cité en pages 159, 165 et 166.)

[Epperson 1987] J.F. Epperson. *On the Runge example.* Amer. Math. Monthly, vol. 94, no. 4, pages 329–341, 1987. (Cité en page 55.)

[Fleischer 2003] M. Fleischer. *The measure of Pareto optima applications to multi-objective metaheuristics.* In Evolutionary Multi-Criterion Optimization, pages 74–74. Springer, 2003. (Cité en pages 21 et 26.)

[Floreano & Mondada 1994] D. Floreano and F. Mondada. *Automatic creation of an autonomous agent : Genetic evolution of a neural-network driven robot.* From animals to animats, vol. 3, pages 421–430, 1994. (Cité en pages 2 et 35.)

[Floreano & Mondada 1998] D. Floreano and F. Mondada. *Evolutionary neurocontrollers for autonomous mobile robots.* Neural Networks, vol. 11, no. 7-8, pages 1461–1478, 1998. (Cité en pages 4, 36, 44, 73 et 129.)

[Floreano & Urzelai 2001] D. Floreano and J. Urzelai. *Evolution of plastic control networks.* Autonomous Robots, vol. 11, no. 3, pages 311–317, 2001. (Cité en pages 2, 39, 44, 45 et 129.)

[Fogel *et al.* 1966] L.J. Fogel, A.J. Owens and M.J. Walsh. Artificial intelligence through simulated evolution. John Wiley & Sons, 1966. (Cité en page 7.)

[Fogel 1995] D.B. Fogel. Evolutionary computation : Toward a new philosophy of machine intelligence. Piscataway, NJ :IEEE Press, 1995. (Cité en page 7.)

[Fogel 2000] D.B. Fogel. *Introduction to evolutionary computation.* Evolutionary Computation 1 : Basic algorithms and operators, vol. 1, page 1, 2000. (Cité en page 7.)

[Fonseca & Fleming 1995] C.M. Fonseca and P.J. Fleming. *An overview of evolutionary algorithms in multiobjective optimization.* Evolutionary computation, vol. 3, no. 1, pages 1–16, 1995. (Cité en pages 3, 9 et 15.)

[Fonseca & Fleming 1998] C.M. Fonseca and P.J. Fleming. *Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation.* Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on, vol. 28, no. 1, pages 26–37, 1998. (Cité en page 29.)

[Fornberg & Zuev 2007] B. Fornberg and J. Zuev. *The Runge phenomenon and spatially variable shape parameters in RBF interpolation.* Computers & Mathematics with Applications, vol. 54, no. 3, pages 379–398, 2007. (Cité en page 55.)

[Fraser 1957a] A.S. Fraser. *Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction.* Australian Journal of Biological Sciences, vol. 10, pages 484–491, 1957. (Cité en page 7.)

[Fraser 1957b] A.S. Fraser. *Simulation of Genetic Systems by Automatic Digital Computers. II. Effects of linkage or rates of advance under selection.* Australian Journal of Biological Sciences, vol. 10, pages 492–499, 1957. (Cité en page 7.)

[Fraser 1958] A.S. Fraser. *Monte Carlo analyses of genetic models.* Nature, vol. 181, pages 208–209, 1958. (Cité en page 7.)

[Fraser 1960] A.S. Fraser. *Simulation of Genetic Systems by Automatic Digital Computers. VI. Epistasis.* Australian Journal of Biological Sciences, vol. 13, no. 2, pages 150–162, 1960. (Cité en page 7.)

[Fu *et al.* 2008] G. Fu, D. Butler and S.T. Khu. *Multiple objective optimal control of integrated urban wastewater systems.* Environmental Modelling & Software, vol. 23, no. 2, pages 225–234, 2008. (Cité en page 61.)

[Funes & Pollack 1999] P. Funes and J. Pollack. *Computer evolution of buildable objects.* Evolutionary design by computers, pages 387–403, 1999. (Cité en pages 2 et 38.)

[Gagné & Schoenauer 2006] C Gagné and M Schoenauer. *Genetic programming, validation sets, and parsimony pressure.* Genetic Programming, 2006. (Cité en pages 43 et 130.)

[Giguere *et al.* 2006] P. Giguere, G. Dudek, C. Prahacs and S. Saunderson. *Environment identification for a running robot using inertial and actuator cues.* Proceedings of Robotics Science and System (RSS 2006), 2006. (Cité en page 172.)

[Giunta *et al.* 1998] A.A. Giunta, L.T. Watson and J. Koehler. *A comparison of approximation modeling techniques : polynomial versus interpolating models.* AIAA paper, pages 98–4758, 1998. (Cité en pages 53 et 54.)

[Goldberg & Deb 1991] D.E. Goldberg and K. Deb. *A comparative analysis of selection schemes used in genetic algorithms.* Foundations of genetic algorithms, vol. 1, pages 69–93, 1991. (Cité en page 13.)

[Goldberg & Richardson 1987] D.E. Goldberg and J. Richardson. *Genetic algorithms with sharing for multimodal function optimization.* In Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, pages 41–49. L. Erlbaum Associates Inc., 1987. (Cité en pages 13 et 32.)

[Gomez 2009] F.J. Gomez. *Sustaining diversity using behavioral information distance.* In Proceedings of GECCO'09, pages 113–120. ACM, 2009. (Cité en pages 32, 34, 131 et 133.)

[Gongora *et al.* 2009] M.A. Gongora, B.N. Passow and A.A. Hopgood. *Robustness analysis of evolutionary controller tuning using real systems.* In Proceedings of IEEE CEC, pages 606–613. Institute of Electrical and Electronics Engineers Inc., The, 2009. (Cité en pages 36, 73 et 74.)

[Grand *et al.* 2004a] C. Grand, F. BenAmar, F. Plumet and P. Bidaud. *Decoupled control of posture and trajectory of the hybrid wheel-legged robot Hylos.* In Proceedings of IEEE ICRA, volume 5, pages 5111–5116, 2004. (Cité en pages 100 et 166.)

[Grand *et al.* 2004b] C. Grand, F. Benamar, F. Plumet and P. Bidaud. *Stability and traction optimization of reconfigurable vehicles . Application to an hybrid wheel-legged robot.* The International Journal of Robotics Research, vol. 23, no. 10-11, 2004. (Cité en pages 48, 159, 165 et 166.)

[Grand *et al.* 2010] C. Grand, F. Benamar and F. Plumet. *Motion kinematics analysis of wheeled-legged rover over 3D surface with posture adaptation.* Mechanism and Machine Theory, vol. 45, no. 3, pages 477–495, 2010. (Cité en page 48.)

[Groşan & Oltean 2004] C. Groşan and M. Oltean. *Improving the performance of evolutionary algorithms for the multiobjective 0/1 knapsack problem using ε-dominance.* Computational Science-ICCS 2004, pages 674–677, 2004. (Cité en page 24.)

[Hancock 1994] P. Hancock. *An empirical comparison of selection methods in evolutionary algorithms.* Evolutionary Computing, pages 80–94, 1994. (Cité en pages 12 et 13.)

[Handl *et al.* 2008] J. Handl, S. Lovell and J. Knowles. *Multiobjectivization by decomposition of scalar cost functions.* Parallel Problem Solving from Nature–PPSN X, pages 31–40, 2008. (Cité en pages 31 et 32.)

[Hansen *et al.* 2009] N. Hansen, S. Finck, R. Ros and A. Auger. *Real-parameter black-box optimization benchmarking 2009 : Noiseless functions definitions.* Rapport technique, INRIA Saclay - Île-de-France, 2009. (Cité en page 62.)

[Hartland & Bredèche 2006] C. Hartland and N. Bredèche. *Evolutionary Robotics, Anticipation and the Reality Gap.* In Proceedings of ROBIO'06, pages 1640–1645, 2006. (Cité en page 39.)

[Harvey *et al.* 1992] I. Harvey, P. Husbands and D. Cliff. *Issues in evolutionary robotics.* In Proceedings of SAB. MIT Press Bradford Books, 1992. (Cité en page 73.)

[Harvey *et al.* 1993] I. Harvey, P. Husbands and D. Cliff. *Issues in evolutionary robotics.* From animals to animats, vol. 2, pages 364–373, 1993. (Cité en pages 1, 29, 32, 35 et 43.)

[Harvey *et al.* 1994] I. Harvey, University of Sussex. School of Cognitive and Computing Sciences. *Seeing the light : Artificial evolution, real vision.* In Proceedings of the rd Intl Conf on Simulation of Adaptive Behaviour SAB. MIT Press, 1994. (Cité en page 30.)

[Hebbel *et al.* 2006] M. Hebbel, R. Kosse and W. Nistico. *Modeling and learning walking gaits of biped robots.* In Proceedings of the Workshop on Humanoid Soccer Robots of the IEEE-RAS International Conference on Humanoid Robots, pages 40–48, 2006. (Cité en pages 2 et 36.)

[Heidrich-Meisner & Igel 2009] V. Heidrich-Meisner and C. Igel. *Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search.* Proceedings of ICML, pages 1–8, 2009. (Cité en page 47.)

[Hemker *et al.* 2006] T. Hemker, H. Sakamoto, M. Stelzer and O. von Stryk. *Hardware-in-the-loop optimization of the walking speed of a humanoid robot.* In Proceedings of CLAWAR, pages 614–623, 2006. (Cité en pages 42, 47, 61, 70, 105 et 125.)

[Hemker *et al.* 2009] T. Hemker, M. Stelzer, O. Von Stryk and H. Sakamoto. *Efficient walking speed optimization of a humanoid robot.* The International Journal of Robotics Research, vol. 28, no. 2, page 303, 2009. (Cité en pages 2, 40, 41, 42, 47, 51, 59, 68, 70, 79 et 155.)

[Holland 1975] J.H. Holland. Adaptation in natural and artificial systems : An introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, 1975. (Cité en page 7.)

[Holland 1992] J.H. Holland. Adaptation in natural and artificial systems. The MIT Press, 1992. (Cité en page 13.)

[Hornby *et al.* 2002] G. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto and M. Fujita. *Evolving robust gaits with AIBO.* In Proceedings of IEEE ICRA, volume 3, pages 3040–3045, 2002. (Cité en pages 36 et 73.)

[Hua *et al.* 2009] Z. Hua, M. Dehai and W. Cheng. *Optimization of the spatial interpolation for groundwater depth in Shule River basin.* In 2009 International Conference on Environmental Science and Information Application Technology, pages 415–418. IEEE, 2009. (Cité en pages 51, 61 et 62.)

[Huang *et al.* 2006] D. Huang, TT Allen, WI Notz and N. Zeng. *Global optimization of stochastic black-box systems via sequential kriging meta-models.* Journal of Global Optimization, vol. 34, no. 3, pages 441–466, 2006. (Cité en page 70.)

[Hughes 2005] E.J. Hughes. *Evolutionary many-objective optimisation : many once or one many ?* In Evolutionary Computation, 2005. The 2005 IEEE Congress on, volume 1, pages 222–227. IEEE, 2005. (Cité en page 27.)

[Hüsken *et al.* 2005] M. Hüsken, Y. Jin and B. Sendhoff. *Structure optimization of neural networks for evolutionary design optimization.* Soft Computing-A Fusion of Foundations, Methodologies and Applications, vol. 9, no. 1, pages 21–28, 2005. (Cité en page 115.)

[Igel *et al.* 2007] C. Igel, N. Hansen and S. Roth. *Covariance matrix adaptation for multi-objective optimization.* Evolutionary Computation, vol. 15, no. 1, pages 1–28, 2007. (Cité en page 26.)

[Ijiri 1965] Y. Ijiri. Management goals and accounting for control. North-Holland Publishing Company, 1965. (Cité en pages 16 et 19.)

[Ishibuchi & Nojima 2006] H. Ishibuchi and Y. Nojima. *Comparison between single-objective and multi-objective genetic algorithms : Performance comparison and performance measures.* In Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, pages 1143–1150. IEEE, 2006. (Cité en page 27.)

[Ishibuchi *et al.* 2008] H. Ishibuchi, N. Tsukamoto and Y. Nojima. *Evolutionary many-objective optimization : A short review.* In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on, pages 2419–2426. IEEE, 2008. (Cité en page 28.)

[Ito & Gofuku 2003] K. Ito and A. Gofuku. *Emergence of adaptive behaviors by redundant robots-Robustness to changes environment and failures.* In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on, volume 4, pages 2572–2579. IEEE, 2003. (Cité en page 172.)

[Jakobi *et al.* 1995] N. Jakobi, P. Husbands and I. Harvey. *Noise and the reality gap : The use of simulation in evolutionary robotics.* Proceedings of ECAL, pages 704–720, 1995. (Cité en pages 2, 35, 37, 38, 44, 45, 73 et 129.)

[Jakobi 1997] N. Jakobi. *Evolutionary robotics and the radical envelope-of-noise hypothesis.* Adaptive behavior, vol. 6, no. 2, page 325, 1997. (Cité en pages 37, 38, 43, 45, 46, 85, 86, 87, 88, 90, 95 et 129.)

[Jakobi 1998a] N. Jakobi. *Minimal Simulations for Evolutionary Robotics.* PhD thesis, University of Sussex, 1998. (Cité en pages 2, 10, 35, 38, 43, 46, 74, 86, 90, 129, 130, 136, 175 et 186.)

[Jakobi 1998b] N. Jakobi. *Running across the reality gap : Octopod locomotion evolved in a minimal simulation.* Lecture Notes in Computer Science, vol. 1468, pages 39–58, 1998. (Cité en pages 30 et 37.)

[Jansen & Wegener 1999] T. Jansen and I. Wegener. *On the analysis of evolutionary algorithms - a proof that crossover really can help.* Algorithms-ESA'99, pages 700–700, 1999. (Cité en pages 10 et 12.)

[Jarrault *et al.* 2010] P. Jarrault, C. Grand and P. Bidaud. *Large Obstacle Clearance Using Kinematic Reconfigurability for a Rover with an Active Suspension.* In Proceedings of CLAWAR'10, pages 1–8, 2010. (Cité en pages 48, 159, 165 et 166.)

[Jensen 2003] M.T. Jensen. *Reducing the run-time complexity of multiobjective EAs : The NSGA-II and other algorithms.* Evolutionary Computation, IEEE Transactions on, vol. 7, no. 5, pages 503–515, 2003. (Cité en page 24.)

[Jensen 2004] M.T. Jensen. *Helper-objectives : Using multi-objective evolutionary algorithms for single-objective optimisation.* Journal of Mathematical Modelling and Algorithms, vol. 3, no. 4, pages 323–347, 2004. (Cité en page 31.)

[Jin *et al.* 2001] R. Jin, W. Chen and T.W. Simpson. *Comparative studies of metamodelling techniques under multiple modelling criteria.* Structural and Multidisciplinary Optimization, vol. 23, no. 1, pages 1–13, 2001. (Cité en page 53.)

[Jin 2005] Y. Jin. *A comprehensive survey of fitness approximation in evolutionary computation.* Soft Computing-A Fusion of Foundations, Methodologies and Applications, vol. 9, no. 1, pages 3–12, 2005. (Cité en pages 41, 51, 52, 53, 54, 59, 61, 68, 69, 79, 80, 93 et 104.)

[Jones *et al.* 1998] D.R. Jones, M. Schonlau and W.J. Welch. *Efficient global optimization of expensive black-box functions.* Journal of Global optimization, vol. 13, no. 4, pages 455–492, 1998. (Cité en pages 51, 62 et 70.)

[Jones 2001] D.R. Jones. *A taxonomy of global optimization methods based on response surfaces.* Journal of Global Optimization, vol. 21, no. 4, pages 345–383, 2001. (Cité en pages 41, 51, 54, 61, 69, 70, 79, 80, 93 et 104.)

[Joukhadar *et al.* 1997] A. Joukhadar, F. Garat and C. Laugier. *Parameter identification for dynamic simulation.* In Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on, volume 3, pages 1928–1933. IEEE, 1997. (Cité en page 162.)

[Keane & Nair 2005] A.J. Keane and P.B. Nair. *Computational approaches for aerospace design.* The Pursuit of Excellence, 2005. (Cité en page 51.)

[Khare *et al.* 2003] V. Khare, X. Yao and K. Deb. *Performance scaling of multi-objective evolutionary algorithms.* In Evolutionary Multi-Criterion Optimization, pages 72–72. Springer, 2003. (Cité en page 27.)

[Khosla & Kanade 1985] P.K. Khosla and T. Kanade. *Parameter identification of robot dynamics.* In Decision and Control, 1985 24th IEEE Conference on, volume 24, pages 1754–1760. IEEE, 1985. (Cité en page 162.)

[Kim *et al.* 2006] S. Kim, C.H. Kim and J.H. Park. *Human-like arm motion generation for humanoid robots using motion capture database.* In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pages 3486–3491. IEEE, 2006. (Cité en pages 51, 52 et 54.)

[Kleiner *et al.* 2006] A. Kleiner, J. Prediger and B. Nebel. *RFID technology-based exploration and SLAM for search and rescue.* In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pages 4054–4059. IEEE, 2006. (Cité en page 171.)

[Knowles & Corne 2002] J. Knowles and D. Corne. *On metrics for comparing nondominated sets.* In Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on, volume 1, pages 711–716. IEEE, 2002. (Cité en page 26.)

[Knowles & Corne 2003] J. Knowles and D. Corne. *Properties of an adaptive archiving algorithm for storing nondominated vectors.* Evolutionary Computation, IEEE Transactions on, vol. 7, no. 2, pages 100–116, 2003. (Cité en page 26.)

[Knowles *et al.* 2001] J. Knowles, R. Watson and D. Corne. *Reducing local optima in single-objective problems by multi-objectivization.* In Evolutionary Multi-Criterion Optimization, pages 269–283. Springer, 2001. (Cité en pages 31 et 32.)

[Kohavi 1995] R. Kohavi. *A study of cross-validation and bootstrap for accuracy estimation and model selection.* In International joint Conference on artificial intelligence, volume 14, pages 1137–1145, 1995. (Cité en page 131.)

[Kohl & Stone 2004] N. Kohl and P. Stone. *Policy gradient reinforcement learning for fast quadrupedal locomotion.* In Proceedings of ICRA, volume 3, pages 2619–2624. IEEE, 2004. (Cité en pages 36, 48, 49, 73, 160 et 173.)

[Kondo *et al.* 1999] T. Kondo, A. Ishiguro, S. Tokura, Y. Uchikawa and P. Eggenberger. *Realization of robust controllers in evolutionary robotics : a dynamically-rearranging neural network approach.* In Proceedings of CEC'99, volume 1. IEEE, 1999. (Cité en pages 45 et 129.)

[Koos & Mouret 2011] S. Koos and J.-B. Mouret. *Online Discovery of Locomotion Modes for Wheel-Legged Hybrid Robots : a Transferability-based Approach.* In Proceedings of CLAWAR (à paraître). World Scientific Publishing Co., 2011. (Cité en pages 165 et 166.)

[Koos *et al.* 2009] S. Koos, J.-B. Mouret and S. Doncieux. *Automatic system identification based on coevolution of models and tests.* In Proceedings of IEEE CEC, pages 119–126, 2009. (Cité en pages 40 et 75.)

[Koos *et al.* 2010] S. Koos, J.-B. Mouret and S. Doncieux. *Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers.* In Proceedings of GECCO'10. ACM, publisher, 2010. (Cité en page 73.)

[Koos *et al.* 2011] S. Koos, J.-B. Mouret and S. Doncieux. *The Transferability Approach : Crossing the Reality Gap in Evolutionary Robotics (à paraître).* IEEE Transactions on Evolutionary Computation, 2011. (Cité en page 73.)

[Kording *et al.* 2007] K.P. Kording, J.B. Tenenbaum and R. Shadmehr. *The dynamics of memory as a consequence of optimal adaptation to a changing body.* Nature neuroscience, vol. 10, no. 6, page 779, 2007. (Cité en page 180.)

[Koziel & Bandler 2007] S. Koziel and J.W. Bandler. *Space-mapping optimization with adaptive surrogate model.* Microwave Theory and Techniques, IEEE Transactions on, vol. 55, no. 3, pages 541–547, 2007. (Cité en page 69.)

[Kozlowski 1998] K. Kozlowski. Modelling and identification in robotics. Springer, 1998. (Cité en page 162.)

[Krige 1981] D.G. Krige. *Lognormal-de Wijsian geostatistics for ore evaluation.* South African Institute of Mining and Metallurgy, Johannesburg, 1981. (Cité en page 59.)

[Kumano *et al.* 2006] T. Kumano, S. Jeong, S. Obayashi, Y. Ito, K. Hatanaka and H. Morino. *Multidisciplinary design optimization of wing shape for a small jet aircraft using kriging model.* AIAA paper, vol. 932, page 2006, 2006. (Cité en pages 40 et 41.)

[Lahanas *et al.* 2003] M. Lahanas, D. Baltas and N. Zamboglou. *A hybrid evolutionary algorithm for multi-objective anatomy-based dose optimization in high-dose-rate brachytherapy.* Physics in medicine and biology, vol. 48, page 399, 2003. (Cité en page 28.)

[Lahanas *et al.* 2004] M. Lahanas, K. Karouzakis, S. Giannouli, R. Mould and D. Baltas. *Inverse planning in brachytherapy : Radium to high dose rate 192 iridium afterloading.* Nowotwory Journal of Oncology, vol. 54, no. 6, pages 547–554, 2004. (Cité en page 28.)

[Laumanns *et al.* 1999] M. Laumanns, G. Rudolph and H.P. Schwefel. Approximating the pareto set : Concepts, diversity issues, and performance assessment. Secretary of the SFB 531, 1999. (Cité en page 26.)

[Laumanns *et al.* 2002] M. Laumanns, L. Thiele, K. Deb and E. Zitzler. *Combining convergence and diversity in evolutionary multiobjective optimization.* Evolutionary computation, vol. 10, no. 3, pages 263–282, 2002. (Cité en pages 24, 25 et 26.)

[Leary *et al.* 2004] S.J. Leary, A. Bhaskar and A.J. Keane. *A derivative based surrogate model for approximating and optimizing the output of an expensive computer simulation.* Journal of Global Optimization, vol. 30, no. 1, pages 39–58, 2004. (Cité en pages 61 et 70.)

[Lehman & Stanley 2008] J. Lehman and K.O. Stanley. *Exploiting open-endedness to solve problems through the search for novelty.* Artificial Life, vol. 11, page 329, 2008. (Cité en pages 15 et 31.)

[Lehman & Stanley 2010] J. Lehman and K.O. Stanley. *Abandoning Objectives : Evolution through the Search for Novelty Alone.* Evolutionary computation, 2010. (Cité en pages 15, 31 et 45.)

[Lenser & Veloso 2004] S. Lenser and M. Veloso. *Classification of robotic sensor streams using non-parametric statistics.* In Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, volume 3, pages 2719–2724. IEEE, 2004. (Cité en page 172.)

[Lhomme-Desages *et al.* 2007] D. Lhomme-Desages, C. Grand and J.C. Guinot. *Trajectory Control of a Four-Wheel Skid-Steering Vehicle over Soft Terrain using a Physical Interaction Model.* In Proceedings of ICRA'07 : IEEE/Int. Conf. on Robotics and Automation, pages 1164 – 1169, 2007. (Cité en pages 37 et 73.)

[Li & Chan Hilton 2007] Y. Li and A.B. Chan Hilton. *Optimal groundwater monitoring design using an ant colony optimization paradigm.* Environmental Modelling & Software, vol. 22, no. 1, pages 110–116, 2007. (Cité en page 58.)

[Lipson & Pollack 2000] H. Lipson and J.B. Pollack. *Automatic design and manufacture of artificial lifeforms.* Nature, vol. 406, no. 974-978, 2000. (Cité en page 36.)

[Liu *et al.* 1998] X. Liu, DW Begg and RJ Fishwick. *Genetic approach to optimal topology/controller design of adaptive structures.* International Journal for

Numerical Methods in Engineering, vol. 41, no. 5, pages 815–830, 1998. (Cité en page 16.)

[Lophaven *et al.* 2002] S. Lophaven, H. Nielsen and J. Sondergaard. *DACE – A Matlab Kriging Toolbox.* Rapport technique, IMM-TR-2002-12, IMM, 2002. (Cité en page 105.)

[Lucet *et al.* 2009] E. Lucet, C. Grand, D. SallÃ© and P. Bidaud. *Dynamic yaw and velocity control of the 6WD skid-steering mobile robot RobuROC6 using sliding mode technique.* In Proceedings of IEEE/IROS : Int. Conf. on Robots and Intelligents Systems, pages 4220–4225, October 2009. (Cité en page 1.)

[Lucet *et al.* 2010] E. Lucet, C. Grand, A.V. Terekhov and P. Bidaud. *Experimental study of a fast mobile robot performing a drift maneuver.* In Proceedings of Clawar'10 : 12th Int. Conf. on Climbing and Walking Robots, Nagoya, Japon, 2010. (Cité en page 20.)

[Mack *et al.* 2007] Y. Mack, T. Goel, W. Shyy and R. Haftka. *Surrogate model-based optimization framework : a case study in aerospace design.* Evolutionary Computation in Dynamic and Uncertain Environments, pages 323–342, 2007. (Cité en page 51.)

[Maes *et al.* 2001] J.H.R. Maes, B.M. Bouwman and J.M.H. Vossen. *Effects of d-amphetamine on the performance of rats in an animal analogue of the A-X Continuous Performance Test.* Journal of Psychopharmacology, vol. 15, no. 1, pages 23–28, 2001. (Cité en pages 134, 138 et 139.)

[Mahfoud 1997] S.W. Mahfoud. Handbook of evolutionary computation, chapitre Niching Methods. Taylor & Francis, 1997. (Cité en pages 13 et 14.)

[Malvić & Dureković 2003] T. Malvić and M. Dureković. *Application of methods : Inverse distance weighting, ordinary Kriging and collocated Co-Kriging in porosity evaluation, and comparison of results on the Benicanci and Stari Gradac fields in Croatia.* Nafta, vol. 54, no. 9, pages 331–340, 2003. (Cité en pages 51, 59, 60, 62 et 80.)

[Matheron 1963] G. Matheron. *Principles of geostatistics.* Economic geology, vol. 58, no. 8, page 1246, 1963. (Cité en page 59.)

[Mauldin 1984] M.L. Mauldin. *Maintaining diversity in genetic search.* In Proceedings of the national conference on artificial intelligence, volume 247, page 250, 1984. (Cité en page 13.)

[Meyer *et al.* 1998] J.A. Meyer, P. Husbands and I. Harvey. *Evolutionary robotics : A survey of applications and problems.* In Evolutionary Robotics, pages 1–21. Springer, 1998. (Cité en pages 29 et 32.)

[Miglino *et al.* 1995] O. Miglino, H.H. Lund and S. Nolfi. *Evolving Mobile Robots in Simulated and Real Environments.* Artificial Life, vol. 2, no. 4, pages 417–434, 1995. (Cité en pages 2, 37, 38, 44, 45 et 129.)

[Miller & Goldberg 1995] B.L. Miller and D.E. Goldberg. *Genetic Algorithms, Tournament Selection, and the E ects of Noise.* Urbana, vol. 51, page 61801, 1995. (Cité en page 12.)

[Minato & Asada 2000] T. Minato and M. Asada. *Environmental change adaptation for mobile robot navigation.* In Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on, volume 3, pages 1859–1864. IEEE, 2000. (Cité en pages 47, 49 et 173.)

[Mitchell 1998] M. Mitchell. An introduction to genetic algorithms. The MIT press, 1998. (Cité en page 7.)

[Modalaldoust 2010] S. Modalaldoust. *Evaluating optimized digital elevation precipitation model using IDW method (Case study : Jam & Riz Watershed of Assaloyeh, Iran).* DESERT (BIABAN), 2010. (Cité en page 57.)

[Molina-Cristóbal *et al.* 2005] A. Molina-Cristóbal, I.A. Griffin, P.J. Fleming and D.H. Owens. *Multiobjective controller design : optimising controller structure with genetic algorithms.* In Proceedings of the 16th IFAC World Congress, 2005. (Cité en pages 3 et 29.)

[Mondada *et al.* 2009] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.C. Zufferey, D. Floreano and A. Martinoli. *The e-puck, a robot designed for education in engineering.* In Proceedings of the 9th conference on autonomous robot systems and competitions, volume 1, pages 59–65, 2009. (Cité en page 86.)

[Moore 1975] G.E. Moore. *Progress in digital integrated electronics.* In Electron Devices Meeting, 1975 International, volume 21, pages 11–13. IEEE, 1975. (Cité en page 3.)

[Moraglio & Kattan 2011] A. Moraglio and A. Kattan. *Geometric generalisation of surrogate model based optimisation to combinatorial spaces.* Evolutionary Computation in Combinatorial Optimization, pages 142–154, 2011. (Cité en page 69.)

[Mouret & Doncieux 2008] J.-B. Mouret and S. Doncieux. *Incremental Evolution of Animats' Behaviors as a Multi-objective Optimization.* In From Animals to Animats 10, volume 5040, pages 210–219. Springer, 2008. (Cité en pages 14, 31 et 32.)

[Mouret & Doncieux 2009a] J.-B. Mouret and S. Doncieux. *Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity.* In IEEE Congress on Evolutionary Computation, 2009 (CEC 2009), 2009. (Cité en pages 14, 106 et 144.)

[Mouret & Doncieux 2009b] J.-B. Mouret and S. Doncieux. *Using Behavioral Exploration Objectives to Solve Deceptive Problems in Neuro-evolution.* In Proceedings of GECCO'09, pages 627–634. ACM, 2009. (Cité en pages 14, 31, 34 et 81.)

[Mouret & Doncieux 2010] J.-B. Mouret and S. Doncieux. *Sferes_ v2 : Evolvin' in the Multi-Core World.* In Proceedings of WCCI-CEC 2010, pages 4079–4086, 2010. (Cité en pages 95 et 139.)

[Mouret & Doncieux 2011] J.-B. Mouret and S. Doncieux. *Encouraging Behavioral Diversity in Evolutionary Robotics : an Empirical Study.* Evolutionary Computation, 2011. (Cité en pages 12, 13, 14, 28, 31, 32, 34, 43, 45, 81, 129, 131, 133, 134, 137, 138, 143, 146, 155, 157 et 182.)

[Mouret 2008] J.-B. Mouret. *Pressions sélectives multiples pour l'évolution de réseaux de neurones destinés à la robotique.* PhD thesis, Université Pierre et Marie Curie (UPMC), 2008. (Cité en page 13.)

[Mouret 2011] J.-B. Mouret. Novelty-based multiobjectivization, pages 139–154. Springer, 2011. (Cité en pages 15, 31 et 32.)

[Mucientes *et al.* 2010] M. Mucientes, J. Alcalá-Fdez, R. Alcalá and J. Casillas. *A case study for learning behaviors in mobile robotics by evolutionary fuzzy systems.* Expert Systems with Applications, vol. 37, no. 2, pages 1471–1493, 2010. (Cité en page 30.)

[Mueller *et al.* 2004] T.G. Mueller, N.B. Mathias, K.K. Cornelius, P.L. Barnhisel and R.I. Shearer. *Map quality for ordinary kriging and inverse distance weighted interpolation.* Soil Science Society of America Journal, vol. 68, no. 6, page 2042, 2004. (Cité en pages 59, 60, 62 et 80.)

[Mullur & Messac 2006] A.A. Mullur and A. Messac. *Metamodeling using extended radial basis functions : a comparative approach.* Engineering with Computers, vol. 21, no. 3, pages 203–217, 2006. (Cité en page 53.)

[Nandasana *et al.* 2003] A.D. Nandasana, A.K. Ray and S.K. Gupta. *Applications of the non-dominated sorting genetic algorithm (NSGA) in chemical reaction engineering.* International Journal of Chemical Reactor Engineering, 2003. (Cité en page 28.)

[Nelson *et al.* 2009] A.L. Nelson, G.J. Barlow and L. Doitsidis. *Fitness functions in evolutionary robotics : A survey and analysis.* Robotics and Autonomous Systems, vol. 57, no. 4, pages 345–370, 2009. (Cité en pages 8, 10, 30 et 31.)

[Nistér *et al.* 2006] D. Nistér, O. Naroditsky and J. Bergen. *Visual odometry for ground vehicle applications.* Journal of Field Robotics, vol. 23, no. 1, pages 3–20, 2006. (Cité en page 171.)

[Nolfi & Floreano 2000] S. Nolfi and D. Floreano. Evolutionary robotics : The biology, intelligence, and technology of self-organizing machines. MIT Press, 2000. (Cité en pages 1 et 29.)

[Nolfi *et al.* 1994] S. Nolfi, D. Floreano, O. Miglino and F. Mondada. *How to evolve autonomous robots : Different approaches in evolutionary robotics.* In Proceedings of Artificial Life IV, pages 190–197. Rodney Brooks and Pattie Maes, 1994. (Cité en pages 2, 30 et 37.)

[Ollion & Doncieux 2011] C. Ollion and S. Doncieux. *Why and How to Measure Exploration in Behavioral Space.* In Proceedings of GECCO'11. ACM, 2011. (Cité en pages 15, 32 et 34.)

[Osyczka & Osyczka 2002] A. Osyczka and A. Osyczka. Evolutionary algorithms for single and multicriteria design optimization. Physica-Verlag Heidelberg, 2002. (Cité en page 9.)

[Panait 2003] Liviu Panait. *Methods for evolving robust programs.* Proceedings of GECCO, 2003. (Cité en page 136.)

[Pelikan *et al.* 2005] M. Pelikan, D.E. Goldberg and S. Tsutsui. *Hierarchical Bayesian optimization algorithm : Toward a new generation of evolutionary algorithms.* In SICE 2003 Annual Conference, volume 3, pages 2738–2743. IEEE, 2005. (Cité en page 30.)

[Pinville & Doncieux 2010] T Pinville and S. Doncieux. *Automatic Synthesis of Working memory neural networks with neuroevolution methods.* In Neurocomp 2010, 2010. (Cité en pages 137 et 138.)

[Pinville *et al.* 2011] T. Pinville, S. Koos, J.-B. Mouret and S. Doncieux. *How to Promote Generalisation in Evolutionary Robotics : the ProGAb Approach.* In Proceedings of GECCO'11. ACM, 2011. (Cité en pages 98 et 129.)

[Pires *et al.* 2004] E.J.S. Pires, P.B. de Moura Oliveira and J.A.T. Machado. *Multi-objective genetic manipulator trajectory planner.* Applications of Evolutionary Computing, pages 219–229, 2004. (Cité en pages 3 et 29.)

[Pollack & Lipson 2000] J.B. Pollack and H. Lipson. *The GOLEM project : Evolving hardware bodies and brains.* eh, page 37, 2000. (Cité en page 36.)

[Pollack *et al.* 2000] J.B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby and R. Watson. *Evolutionary techniques in physical robotics.* In Proceedings of ICES, pages 175–186. Springer Verlag, 2000. (Cité en page 35.)

[Prieto *et al.* 2010] A. Prieto, F. Bellas, J. Becerra, B. Priego and R. Duro. *Self-organizing Robot Teams Using Asynchronous Situated Co-evolution.* In Proceedings of SAB, volume 6226 of *Lecture Notes in Computer Science*, pages 565–574. Springer Berlin / Heidelberg, 2010. (Cité en page 74.)

[Quagliarella & Vicini 1997] D. Quagliarella and A. Vicini. *Coupling genetic algorithms and gradient based optimization techniques.* Genetic Algorithms and Evolution Strategy in Engineering and Computer Science - Recent advances and industrial applications, pages 289–309, 1997. (Cité en pages 16 et 18.)

[Queipo *et al.* 2005] N.V. Queipo, R.T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan and P. Kevin Tucker. *Surrogate-based analysis and optimization.* Progress in Aerospace Sciences, vol. 41, no. 1, pages 1–28, 2005. (Cité en pages 51 et 68.)

[Ranjithan *et al.* 1992] S. Ranjithan, JW Eheart and JC Liebman. *Incorporating fixed-cost component of pumping into stochastic groundwater management : A genetic algorithm-based optimization approach.* Eos Transactions AGU, vol. 74, page 125, 1992. (Cité en pages 16 et 18.)

[Ratle 1998] A Ratle. *Accelerating the convergence of evolutionary algorithms by fitness landscape approximation.* PPSN, 1998. (Cité en page 47.)

[Ray 2004] T. Ray. *Applications of multi-objective evolutionary algorithms in engineering design.* Applications of multi-objective evolutionary algorithms, vol. 1, page 29, 2004. (Cité en page 28.)

[Rechenberg 1973] I. Rechenberg. *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Fromman-Holzboog.* Stuttgart. German, 1973. (Cité en page 7.)

[Reed *et al.* 2007] P. Reed, J.B. Kollat and VK Devireddy. *Using interactive archives in evolutionary multiobjective optimization : A case study for long-term groundwater monitoring design.* Environmental Modelling & Software, vol. 22, no. 5, pages 683–692, 2007. (Cité en page 28.)

[Regan *et al.* 2006] W. Regan, F. van Breugel and H. Lipson. *Towards evolvable hovering flight on a physical ornithopter.* In Proceedings of Artificial Life X, volume 100, page 241, 2006. (Cité en pages 36 et 73.)

[Reynolds 1994] C.W. Reynolds. *Evolution of corridor following behavior in a noisy world.* From animals to animats, vol. 3, pages 402–410, 1994. (Cité en pages 44 et 129.)

[Richardson *et al.* 1989] J.T. Richardson, M.R. Palmer, G.E. Liepins and M.R. Hilliard. *Some guidelines for genetic algorithms with penalty functions.* In Proceedings of the 3rd International Conference on Genetic Algorithms, pages 191–197. Morgan Kaufmann Publishers Inc., 1989. (Cité en page 28.)

[Rudolph 1996] G. Rudolph. *Convergence of evolutionary algorithms in general search spaces.* In Evolutionary Computation, 1996., Proceedings of IEEE International Conference on, pages 50–54. IEEE, 1996. (Cité en page 13.)

[Runge 1901] C. Runge. *Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten.* Zeitschrift für Mathematik und Physik, vol. 46, no. 224-243, page 20, 1901. (Cité en page 55.)

[Ruppin 2002] E. Ruppin. *Evolutionary autonomous agents : A neuroscience perspective.* Nature Reviews Neuroscience, vol. 3, no. 2, pages 132–141, 2002. (Cité en pages 4 et 36.)

[Rylatt & Czarnecki 2000] R.M. Rylatt and C.A. Czarnecki. *Embedding Connectionist Autonomous Agents in Time : The 'Road Sign Problem'.* Neural Processing Letters, vol. 12, no. 2, page 145, 2000. (Cité en page 138.)

[Sacks *et al.* 1989] J. Sacks, W.J. Welch, T.J. Mitchell and H.P. Wynn. *Design and analysis of computer experiments.* Statistical science, pages 409–423, 1989. (Cité en page 51.)

[Sasena 2002] M.J. Sasena. *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations.* PhD thesis, University of Michigan, 2002. (Cité en pages 60, 68, 69 et 70.)

[Saunders *et al.* 2011] F. Saunders, E. Golden, R.D. White and J. Rife. *Experimental verification of soft-robot gaits evolved using a lumped dynamic model.* Robotica, vol. 1, no. 1, pages 1–8, 2011. (Cité en pages 2 et 37.)

[Schaal & Atkeson 1994] S. Schaal and C.G. Atkeson. *Robot juggling : Implementation of memory-based learning.* Control Systems Magazine, IEEE, vol. 14, no. 1, pages 57–71, 1994. (Cité en pages 48 et 49.)

[Schaller 1997] R.R. Schaller. *Moore's law : past, present and future.* Spectrum, IEEE, vol. 34, no. 6, pages 52–59, 1997. (Cité en page 3.)

[Schaul *et al.* 2011] T. Schaul, Y. Sun, D. Wierstra, F. Gomez and J. Schmidhuber. *Curiosity-Driven Optimization.* Proceedings of the IEEE Congress on Evolutionary Computation (à paraître), 2011. (Cité en page 70.)

[Schwefel 1981] Hans-Paul Schwefel. Numerical optimization of computer models. Wiley, Chichester, 1981. (Cité en page 7.)

[Sciavicco & Siciliano 2000] L. Sciavicco and B. Siciliano. Modelling and control of robot manipulators. Springer Verlag, 2000. (Cité en page 1.)

[Shepard 1968] D. Shepard. *A two-dimensional interpolation function for irregularly-spaced data.* In Proceedings of the 1968 23rd ACM national conference, pages 517–524. ACM, 1968. (Cité en pages 54, 57, 62, 80, 132 et 164.)

[Simpson *et al.* 2001] T.W. Simpson, T.M. Mauery, J.J. Korte and F. Mistree. *Kriging models for global approximation in simulation-based multidisciplinary design optimization.* AIAA journal, vol. 39, no. 12, pages 2233–2241, 2001. (Cité en pages 51, 53, 54 et 68.)

[Smith 2005] R. Smith. *Open dynamics engine*, 2005. `http://www.ode.org`. (Cité en pages 41 et 167.)

[Soenario & Sluiter 2010] I. Soenario and R. Sluiter. *Optimization of Rainfall Interpolation.* Rapport technique, Koninklijk Nederlands Meteorologisch Instituut, 2010. (Cité en pages 59 et 61.)

[Spall 2003] J.C. Spall. Introduction to stochastic search and optimization : estimation, simulation, and control, volume 64. LibreDigital, 2003. (Cité en pages 3 et 9.)

[Sproewitz *et al.* 2008] A. Sproewitz, R. Moeckel, J. Maye and A. J. Ijspeert. *Learning to Move in Modular Robots using Central Pattern Generators and Online Optimization.* The International Journal of Robotics Research, vol. 27, no. 3-4, pages 423–443, 2008. (Cité en pages 49 et 160.)

[Srinivas & Deb 1994] N. Srinivas and K. Deb. *Multiobjective optimization using nondominated sorting in genetic algorithms.* Evolutionary computation, vol. 2, no. 3, pages 221–248, 1994. (Cité en page 32.)

[Stanley & Miikkulainen 2002] K.O. Stanley and R. Miikkulainen. *Evolving neural networks through augmenting topologies.* Evolutionary computation, vol. 10, no. 2, pages 99–127, 2002. (Cité en pages 10, 12, 32, 34, 137, 154 et 156.)

[Steingrube *et al.* 2010] S. Steingrube, M. Timme, F. Wörgötter and P. Manoonpong. *Self-organized adaptation of a simple neural circuit enables complex robot behaviour.* Nature Physics, vol. 6, no. 3, pages 224–230, 2010. (Cité en pages 48 et 159.)

[Storn & Price 1995] R. Storn and K. Price. *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces*. International Computer Science Institute-Publications-TR, 1995. (Cité en page 62.)

[Terekhov *et al.* 2010a] A.V. Terekhov, J.-B. Mouret and C. Grand. *Stochastic multi-objective optimization for aggressive maneuver trajectory planning on loose surface*. In Proceedings of IFAC : the 7th Symposium on Intelligent Autonomous Vehicles, 2010. (Cité en pages 37 et 73.)

[Terekhov *et al.* 2010b] A.V. Terekhov, J.-B. Mouret and C. Grand. *Stochastic optimization of a neural network-based controller for aggressive maneuvers on loose surfaces*. In Proceedings of IEEE IROS, 2010. (Cité en page 74.)

[Teytaud 2007] O. Teytaud. *On the hardness of offline multi-objective optimization*. Evolutionary Computation, vol. 15, no. 4, pages 475–491, 2007. (Cité en pages 27 et 28.)

[Thierens & Goldberg 1994] D. Thierens and D. Goldberg. *Convergence models of genetic algorithm selection schemes*. Parallel Problem Solving from Nature - PPSN III, pages 119–129, 1994. (Cité en page 13.)

[Thompson *et al.* 1999] A. Thompson, P. Layzell and R.S. Zebulum. *Explorations in design space : Unconventional electronics design through artificial evolution*. IEEE Transactions on Evolutionary Computation, vol. 3, no. 3, pages 167–196, 1999. (Cité en page 38.)

[Toffolo & Benini 2003] A. Toffolo and E. Benini. *Genetic diversity as an objective in multi-objective evolutionary algorithms*. Evolutionary Computation, vol. 11, no. 2, pages 151–167, 2003. (Cité en pages 13, 14 et 32.)

[Trujillo *et al.* 2008] L. Trujillo, G. Olague, E. Lutton and F. Fernández de Vega. *Discovering several robot behaviors through speciation*. Applications of Evolutionary Computing, pages 164–174, 2008. (Cité en pages 32, 34, 131 et 133.)

[Tvrdik & Krivy 1999] J. Tvrdik and I. Krivy. *Simple evolutionary heuristics for global optimization*. Computational statistics & data analysis, vol. 30, no. 3, pages 345–352, 1999. (Cité en page 62.)

[Ulam *et al.* 2005] P. Ulam, A. Goel, J. Jones and W. Murdock. *Using model-based reflection to guide reinforcement learning*. Reasoning, Representation, and Learning in Computer Games, page 107, 2005. (Cité en page 181.)

[Ursem 2002] R. Ursem. *Diversity-guided evolutionary algorithms*. Parallel Problem Solving from Nature - PPSN VII, pages 462–471, 2002. (Cité en pages 13 et 32.)

[Urzelai & Floreano 2001] J. Urzelai and D. Floreano. *Evolution of adaptive synapses : Robots with fast adaptive behavior in new environments*. Evolutionary Computation, vol. 9, no. 4, pages 495–524, 2001. (Cité en pages 2, 39, 44 et 129.)

[Voutchkov & Keane 2006] I. Voutchkov and A.J. Keane. Multiobjective optimization using surrogates. Springer, 2006. (Cité en page 79.)

[Voutchkov & Keane 2010] I. Voutchkov and A.J. Keane. *Multi-objective optimization using surrogates.* Computational Intelligence in Optimization, pages 155–175, 2010. (Cité en page 70.)

[Weingarten *et al.* 2004] J.D. Weingarten, G.A.D. Lopes, M. Buehler, R.E. Groff and D.E. Koditschek. *Automated gait adaptation for legged robots.* In Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, volume 3, pages 2153–2158. IEEE, 2004. (Cité en pages 37, 48, 49 et 73.)

[Wettergreen & Thorpe 1992] D. Wettergreen and C. Thorpe. *Gait generation for legged robots.* In Conference on Intelligent Robots and Systems, 1992. (Cité en page 172.)

[While *et al.* 2006] L. While, P. Hingston, L. Barone and S. Huband. *A faster algorithm for calculating hypervolume.* Evolutionary Computation, IEEE Transactions on, vol. 10, no. 1, pages 29–38, 2006. (Cité en page 26.)

[Whitley 2001] D. Whitley. *An overview of evolutionary algorithms : practical issues and common pitfalls.* Information and software technology, vol. 43, no. 14, pages 817–831, 2001. (Cité en pages 12 et 13.)

[Wilson & Macleod 1993] PB Wilson and MD Macleod. *Low implementation cost IIR digital filter design using genetic algorithms.* In IEE/IEEE Workshop on Natural Algorithms in Signal Processing, volume 1, page 4, 1993. (Cité en page 16.)

[Witteveen & Bijl 2009] J.A.S. Witteveen and H. Bijl. *Explicit mesh deformation using inverse distance weighting interpolation*, 2009. (Cité en page 62.)

[Wolpert & Macready 1997] D.H. Wolpert and W.G. Macready. *No free lunch theorems for optimization.* Evolutionary Computation, IEEE Transactions on, vol. 1, no. 1, pages 67–82, 1997. (Cité en page 9.)

[Yang & Gen 1994] X. Yang and M. Gen. *Evolution program for bicriteria transportation problem.* Computers & industrial engineering, vol. 27, no. 1-4, pages 481–484, 1994. (Cité en page 16.)

[Yang 2005] J.M. Yang. *Gait synthesis for hexapod robots with a locked joint failure.* Robotica, vol. 23, no. 6, pages 701–708, 2005. (Cité en page 172.)

[Yosinski *et al.* 2011] J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J.C. Zagal and H. Lipson. *Generating Gaits for Physical Quadruped Robots : Evolved Neural Networks Vs. Local Parameterized Search.* Proceedings of GECCO'11, 2011. (à paraître). (Cité en pages 2, 4, 36 et 73.)

[Zagal & Lipson 2011] J.C. Zagal and H. Lipson. *Towards Self-Reflecting Machines : Two-Minds in One Robot.* Advances in Artificial Life. Darwin Meets von Neumann, pages 156–164, 2011. (Cité en page 179.)

[Zagal & Ruiz-Del-Solar 2007] J.C. Zagal and J. Ruiz-Del-Solar. *Combining simulation and reality in evolutionary robotics.* Journal of Intelligent and Robotic Systems, vol. 50, no. 1, pages 19–39, 2007. (Cité en pages 40 et 41.)

[Zagal *et al.* 2008] J.C. Zagal, J. Ruiz-del Solar and A.G. Palacios. *Fitness based identification of a robot structure.* Artificial Life, vol. 11, page 733, 2008. (Cité en pages 40, 41, 75 et 180.)

[Zagal *et al.* 2009] J.C. Zagal, J. Delpiano and J. Ruiz-del Solar. *Self-modeling in humanoid soccer robots.* Robotics and Autonomous Systems, vol. 57, no. 8, pages 819–827, 2009. (Cité en pages 40, 41, 75 et 162.)

[Ziemke & Thieme 2002] T. Ziemke and M Thieme. *Neuromodulation of Reactive Sensorimotor Mappings as a Short-Term Memory Mechanism in Delayed Response Tasks.* Adaptive Behavior, vol. 10, no. 3-4, 2002. (Cité en pages 43, 45, 129, 130 et 138.)

[Zitzler & Thiele 1998a] E. Zitzler and L. Thiele. *An evolutionary algorithm for multiobjective optimization : The strength Pareto approach.* Rapport technique, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, 1998. (Cité en page 21.)

[Zitzler & Thiele 1998b] E. Zitzler and L. Thiele. *Multiobjective optimization using evolutionary algorithms-A comparative case study.* In Parallel problem solving from nature-PPSN V, pages 292–301. Springer, 1998. (Cité en page 26.)

[Zitzler *et al.* 2000] E. Zitzler, K. Deb and L. Thiele. *Comparison of multiobjective evolutionary algorithms : Empirical results.* Evolutionary computation, vol. 8, pages 173–195, 2000. (Cité en pages 4 et 36.)

[Zitzler *et al.* 2001] E. Zitzler, M. Laumanns and L. Thiele. *SPEA2 : Improving the strength Pareto evolutionary algorithm.* In EUROGEN, volume 3242, pages 1–21, 2001. (Cité en pages 23 et 24.)

[Zykov *et al.* 2004] V. Zykov, J. Bongard and H. Lipson. *Evolving dynamic gaits on a physical robot.* In Proceedings of GECCO, volume 4, 2004. (Cité en pages 4, 36 et 73.)

Annexe A

Valeurs de p-values obtenues pour les applications du chapitre 2

A.1 Application I : navigation dans un labyrinthe

Table A.1 – Significativité de la comparaison entre chaque couple d'approches d'après un test de Welch.

	Train_ only	*Eval_ all*	*FRI*	*RPG*	*ProGAb*
Train_ only	1	$6.09 \cdot 10^{-7}$	$1.67 \cdot 10^{-5}$	$9.70 \cdot 10^{-4}$	$1.77 \cdot 10^{-12}$
Eval_ all		1	$1.46 \cdot 10^{-6}$	$1.24 \cdot 10^{-5}$	$2.91 \cdot 10^{-9}$
FRI			1	0.147	$1.25 \cdot 10^{-4}$
RPG				1	0.0387
ProGAb					1

A.2 Application II : collecte de balles dans une arène

Table A.2 – Significativité de la comparaison entre chaque couple d'approches d'après un test de Welch.

	Train_ only	*Eval_ all*	*FRI*	*RPG*	*ProGAb*
Train_ only	1	0.24	$4.76 \cdot 10^{-3}$	$3.38 \cdot 10^{-4}$	$1.25 \cdot 10^{-7}$
Eval_ all		1	0.15	0.040	$1.10 \cdot 10^{-4}$
FRI			1	0.42	0.011
RPG				1	0.018
ProGAb					1